# MULTIPROCESSOR ALGORITHMS FOR GENERALIZED NETWORK FLOWS

by

R.H. Clark
and
R.R. Meyer

Computer Sciences Technical Report #739

December 1987

# MULTIPROCESSOR ALGORITHMS FOR GENERALIZED NETWORK FLOWS*

R.H. Clark and R.R. Meyer

Computer Sciences Department and Center for the Mathematical Sciences
The University of Wisconsin-Madison
Madison, Wisconsin 53706 USA

Abstract

This paper presents parallel algorithms for the solution of generalized network optimization problems on a shared-memory multiprocessor. These algorithms exploit the quasi-tree forest basis structure of generalized networks by attempting to perform multiple simplex pivot operations in parallel on disconnected subtrees. We consider algorithms for both single-period generalized networks and multi-period generalized networks. In the latter case, the multi-period structure is utilized in the initial stage of the algorithms in order to initially partition the problem among processors. Computational experience on the Sequent Balance 21000 multiprocessor is presented that demonstrates linear and sometimes superlinear speedup for a large class of test problems.

* This research was supported in part by
NSF grant CCR-8709952 and AFOSR grant AFOSR-86-0194.

## 1. Introduction

In this paper we discuss two distributed algorithms for the solution of generalized network flow problems. These problems are of the following form:

$$\begin{aligned} \min \quad & cx \\ s.t. \quad & Ax = b \qquad\qquad (\mathrm{GP}) \\ & 0 \leq x \leq u \end{aligned}$$

where the matrix $A \in R^{m \times n}$ has no more than two non-zero elements in each column.

In order to pose GP in a form resembling a regular network flow problem, the variables can be scaled so that columns of $A$ containing two non-zero elements are such that one of the elements is 1, and the other non-zero entry is then interpreted as a *flow multiplier*. If the flow multiplier is -1, then the arc is a regular network arc. In general though, the flow multiplier can be positive or negative and can have magnitude equal to 1, less than 1, or greater than 1. If the flow multiplier for a column is in the range $(0, -1)$, the column corresponds to an arc which loses flow. If the flow multiplier is in the range $(-1, -\infty)$ then the column corresponds to an arc which gains flow. A column which has just one nonzero element corresponds to a *root-arc*, an arc which is incident to just one node. Generalized network flow problems have applications in the areas of scheduling, cash management and production and are discussed in [Glover, et al., 73] and [Glover, et al., 78]. A specialization of the primal simplex algorithm for GP is described in [Jensen and Barnes 80] and [Kennington and Helgason 80]. As with the regular network flow problem, which we designate as NP, the simplex algorithm for GP can be executed on a graph. One difference between GP and NP is that the graph of any basis for NP consists of one rooted tree, while the graph of a basis for GP is a forest of quasi-trees. A quasi-tree is a tree with one additional arc, making it either a rooted tree or a tree with exactly one cycle. Figure 1.1 shows a forest of quasi-trees. In [Engquist and Chang 85] data structures for GP are discussed which are primarily based on those of [Adolphson 82] and [Barr, Glover and Klingman 79]. These data structures were used to implement GRNET [Engquist and Chang 85], a sequential version of the primal simplex algorithm for GP. [Engquist and Chang 85] establishes that on a CYBER 170/750, GRNET is about 50 times faster than MINOS [Murtagh and Saunders 78], a standard LP code, on problems of the form GP.

The distributed algorithms for GP discussed below are made possible by the disconnected nature of the basis graph as described above. Executing a pivot involves updating only one or two quasi-trees in the basis. Therefore, two or more processors can execute pivots simultaneously, as long as they operate on different quasi-trees. A description of how this approach was used to develop PAGEN-K, a distributed version of GRNET, can be found in [Chang, et al., 87]. PAGEN-K uses a user specified number $k$ of processors.

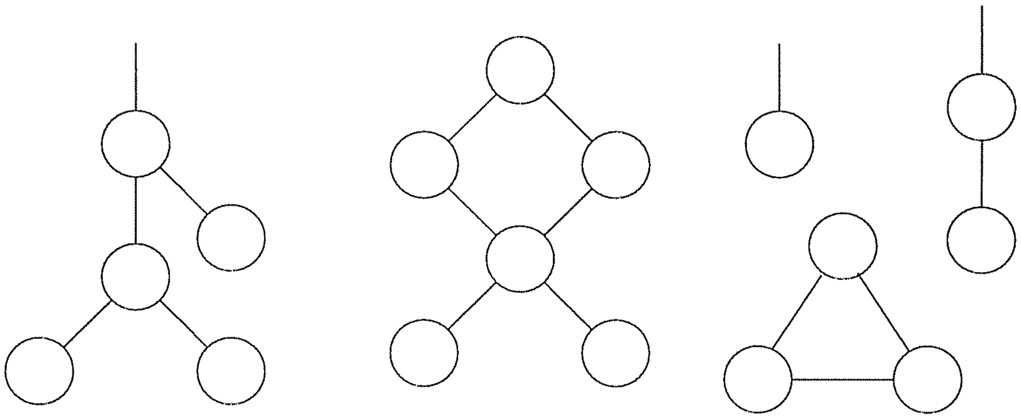

Figure 1.1  A Forest of Quasi-Trees

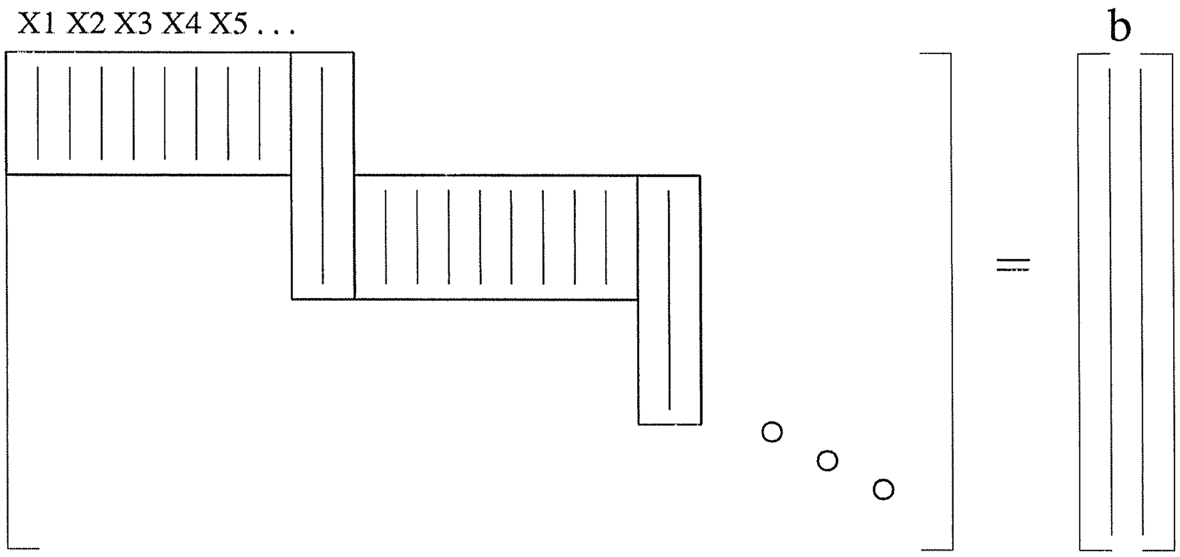

Figure 1.2  Multi-period Constraints

While one processor, the "master" processor, computes the reduced costs on arcs between nodes belonging to different processors, the remaining $(k-1)$ processors compute reduced costs on "local" arcs belonging to the trees assigned to them and execute pivots. Some impressive results were reported in [Chang, et al., 87] for a group of multi-period problems for which the matrix $A$ has the staircase structure given in figure 1.2. The results, however, neglect the communication time between processors.

In our work, we are using a tightly coupled system, the Sequent Balance 21000. This is a time sharing system which permits the user to select the number of processors to be used. Sequent provides a parallel programming library which includes commands that fork processes and coordinate processors. Communication time on this machine is essentially negligible because all processors access a common memory. However, there is an overhead associated with references to memory because there is just one bus connecting the processors to the memory, and processors must take turns using the bus. This can hurt performance. Also, any algorithm for GP written for this machine must ensure that only one processor updates a quasi-tree at any given time. Otherwise, the tree functions may not be updated correctly. To ensure mutual exclusion from quasi-trees, we *lock* the quasi-trees. This is done with hardware locks which use an atomic write operation to set a flag in a boolean array. Processors incur an overhead cost when locking and unlocking quasi-trees, and this can also hurt performance.

Despite the bus bottleneck and the cost of locking and unlocking, the shared memory machine seems to be an appropriate architecture for the solution of generalized network flow problems. We have developed a distributed version of GRNET called PGRNET, which solves a range of generalized network flow problems with a linear and sometimes superlinear speedup. This algorithm and the corresponding computational results are presented in section 2. Our second algorithm, MPGRNET, is a variant of PGRNET for multi-period problems. This second algorithm, described in section 3, uses PGRNET late in the solution process, but does most of the pivoting in a stage in which processors are allocated a particular set of quasi-trees and are permitted to execute pivots only on those quasi-trees. This mimics the loosely coupled environment, and eliminates the need for locking quasi-trees in that stage.

In addition to the investigation of issues related to parallelism, we introduce below a heuristic for candidates for pivoting in the simplex method that reduces computing time by as much as 45% in the test problems considered.

## 2. PGRNET, a distributed version of GRNET

### 2.1 The Algorithm

In this section we describe the sequential program GRNET and our parallel version PGRNET. A rough flow chart of both will be given. In section 2.2, we will discuss a heuristic which we have added to GRNET, and we will describe computational results.

GRNET uses an artificial starting basis discussed in [Glover, et al., 74]. An artificial root arc with a high (*bigM*) cost is attached to each node forming a quasi-tree with just one node and one arc. An example starting basis is shown in figure 2.0. Each artificial arc is given a flow that satisfies the demand of the corresponding node. After the starting basis has been generated, GRNET solves the problem in two main stages. In STAGE 1, the processor makes a candidate list of size *listsize* containing pivot eligible arcs. This is done by sweeping through the list of arcs and adding pivot eligible arcs to the list when they are found. When the list has *listsize* entries, the processor can begin pivoting. To find a pivot arc during STAGE 1, the processor looks through its candidate list to find the pivot eligible arc with the greatest reduced cost in absolute value. The processor executes the pivot, removes the arc from the list and looks for another pivot eligible arc in the candidate list. If there are no pivot eligible arcs in the list, or if the list has fewer than (*listsize*/2) entries, a new list is made. If, in the process of making a new candidate list, fewer than (*listsize*/2) pivot eligible arcs are found in the entire arc list, STAGE 2 is begun. In STAGE 2, optimality is achieved by sweeping through the list of arcs and pivoting on any that are pivot eligible. Optimality is verified when the processor can sweep through all arcs, finding none that are pivot eligible.

In developing a distributed version of GRNET, we tried a number of different strategies. The best results came from what we call an *arc ownership* algorithm. Processors are given a specific subset of the arc list (rather than a subset of the quasi-trees), and each processor selects pivot arcs from that subset and executes the pivots. Arcs are divided evenly between processors. If there are $m$ arcs and $P$ processors, then processor 1 gets arcs (1) through $(m/P)$, processor 2 gets arcs $(m/P+1)$ through $(2m/P)$ and so forth. Optimality is reached when all processors make a sweep through their arcs finding none that are pivot eligible. The dual variables of all the nodes, the predecessor threads, the successor threads and all other tree functions are stored in the shared memory and are available to all processors. The program that each processor executes is almost identical to GRNET. During a *parallel pivoting* stage, STAGE 1, each processor makes a candidate list of pivot eligible arcs. The candidate lists are made in the same way the the (single) candidate list is made in GRNET, except that the individual processors look only through their own subset of the arc list to find pivot eligible arcs. A processor $p$ choses from its candidate list the pivot eligible arc which has the greatest reduced cost in absolute value. If the quasi-trees at the ends of the arc have not been locked by another processor, $p$ locks the quasi-trees, performs the pivot and removes the arc from the candidate list. If the quasi-trees are already locked, processor $p$ removes the arc from the candidate list and choses another arc. When half of the arcs in the candidate list have been removed, a new candidate list is made. If the new candidate list has $listsize/2$ or fewer entries, STAGE 2 is begun.

STAGE 2 corresponds to the *verification of optimality* stage in GRNET. Optimality is achieved by performing any remaining pivots. Processors sweep through their arc lists looking for pivot eligible arcs. If processor $p$ finds one, it locks the quasi-trees at either end of the arc, executes the pivot, and interrupts the other processors. The interrupt is not an interrupt in the usual sense. It involves setting a flag in a shared array, and all processors check this array frequently during STAGE 2. The interrupt mechanism is needed because a pivot executed by $p$ might cause an arc owned by another processor to become pivot eligible. When the other processors find that they have been interrupted, they restart their sweep. If $p$ finds that one of the trees at the ends of a pivot eligible arc is locked, it simply interrupts the other processors and continues its sweep.

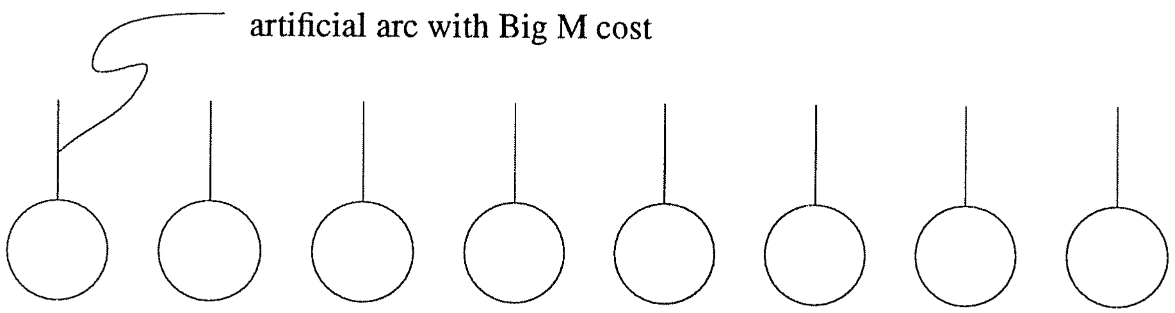

Figure 2.0 Artificial Starting Basis for GRNET

In summary, the (sequential) GRNET algorithm is:

INITIALIZATION
The processor generates an initial feasible solution to a big M problem and proceeds to STAGE 1.

STAGE 1 (*pivoting*)
The processor scans the arc list to make a candidate list. Pivot arcs are chosen from the candidate list and pivots are executed. When it is not possible to make a candidate list with more than (*listsize*/2) entries, the processor proceeds to STAGE 2.

STAGE 2 (*verification of optimality*)
The processor scans the arc list to find any remaining pivot eligible arcs. If it finds a pivot eligible arc, it performs the pivot. If a complete sweep through the arc list can be made without finding any pivot eligible arcs, optimality has been reached.

The (parallel) PGRNET algorithm is:

INITIALIZATION
Processor 1 generates an initial feasible solution to a big M problem.

STAGE 1 (*parallel pivoting*)
Processors scan their arc lists to develop candidate lists. Pivot arcs are chosen from the candidate lists, and quasi-trees are locked before pivots are made. When a processor cannot develop a candidate list with more than (*listsize*/2) entries, that processor proceeds to Stage 2.

STAGE 2 (*verification of optimality*)
Processors scan their arc lists simultaneously to locate any remaining pivot eligible arcs. If a processor finds a pivot eligible arc, it locks the trees involved, performs the pivot and interrupts the other processors. If all processors make a sweep through their lists without being interrupted and without finding any pivot eligible arcs, optimality has been reached.

## 2.2 Pricing Heuristics and Granularity

The sequential program GRNET uses a pivoting strategy due to [Mulvey 78]. This involves periodically developing a list of pivot eligible arcs called a candidate list and choosing pivot arcs from this list on the basis of their reduced costs. We have added a heuristic to GRNET which allows the list length, or list size, to start at 2, and then grow if necessary to a maximum size of 32. We say that a problem is *large-grained* if the quasi-trees in the basis are large, and *small-grained* otherwise. If the problem being solved is very large-grained, the maximum list size of 32 will generally be attained. We have found that GRNET solves small-grained problems more quickly with a short candidate list.

The initial starting basis is very small-grained because each quasi-tree contains only one arc. In this situation, candidate lists of pivot eligible arcs should be relatively short. Our heuristic for tuning the size of the candidate list starts the list size at 2. If a pivot unites two quasi-trees, the following code is executed, and may increase *listsize*, the size of the list. (Parameters in this section of code are discussed below.)

```
$measure = 275 * (T(root)/nodes \cdot per \cdot period)$
if (($measure$.gt.$listsize$) .and. ($listsize$.lt.32)) then
    $listsize = listsize + 2$
endif
```

The number of nodes in the newly formed quasi-tree is $T(root)$. The number $nodes \cdot per \cdot period$ is equal to the number of nodes $n$ in the problem, unless the problem is a multi-period problem. In this case $nodes \cdot per \cdot period$ gives the average number of nodes per period. The code computes the relative size of the newly formed quasi-tree and increments *listsize* if the quasi-tree is sufficiently large. The list size never increases by more than two during a pivot. This means that it becomes large only if many pivots are executed which coalesce large quasi-trees. The results below show the effects of using this variable list size heuristic compared to a fixed list size of 32.

The most significant factor in the efficiency of our algorithm is the granularity of the problem being solved. If the number of quasi-trees in the basis is small, it is less likely that a processor $p$ will find that the quasi-trees at the ends of a pivot arc are not locked. If they are locked, $p$ removes the arc from the list and looks look for another in its candidate list. Thus when the number of quasi-trees in the basis is small, processors are likely to reject "best" arcs chosen from their candidate lists, and this tends to increase the total number of pivots required to reach optimality. Candidate lists are also exhausted quickly because many arcs that are chosen turn out to connect locked quasi-trees. This increases the total amount of time that is spent making candidate lists. A problem which had only one quasi-tree in the optimal solution might be solved only slightly faster by PGRNET than by GRNET.

The performance of PGRNET is also degraded when nodes are distributed unevenly between the quasi-trees. A problem can be small-grained in the sense that it involves a large number of quasi-trees, but if most of the nodes belong to just one quasi-tree, that quasi-tree will be locked most of the time. In this case, the processors compete for access to one quasi-tree. Candidate lists are exhausted quickly because most pivot eligible arcs have an end in this quasi-tree.

Alternatives to rejecting arcs with an end in a locked quasi-tree might be to store them in a temporary stack, or simply to wait until the quasi-tree becomes available. Our experience indicates that these alternatives are less efficient than the algorithm we are using. In section 4 we discuss research directions for problems having a small number of quasi-trees in the basis.

We measure the performance of PGRNET by calculating the speedup for various problems. For a given problem, we define the speedup for $P$ processors in the following way:

$$\text{speedup(P)} = \frac{\text{CPU time required by one processor}}{\text{CPU time required by } P \text{ processors}}$$

The DYNIX operating system on the Sequent Balance 21000 provides the user time and the system time needed to execute a command. Due to the multi-processor nature of DYNIX, it is possible for the CPU time (user + system time) to greatly exceed the real time. Also, the system time reported can depend on the number of users on the system. For these reasons, we report only the user time as the CPU time.

Our goal is to achieve "linear" speedup for all test problems. In other words, we would like the speedup on $P$ processors to equal $P$, regardless of the nature of the problem being solved. If the speedup is greater than $P$ for a given problem, we say that the speedup is *superlinear*, and if the speedup is less than $P$, we say it's *sublinear*. A sublinear speedup is undesirable because it indicates that processors are not being used as efficiently as possible. A superlinear speedup can indicate that improvement is needed in the sequential program. PGRNET exhibits superlinear speedup for a number of problems discussed below, because it does fewer pivots than the sequential version in solving these problems. This means that PGRNET selects a better sequence of pivots than GRNET. Since PGRNET selects pivots according to the magnitude of their reduced costs and according to the availability of the associated quasi-trees, it's hard to determine how one might adapt the pivot selection in GRNET to get the same sequence of pivots as PGRNET. The occasional unavailability of quasi-trees is something that would be difficult to incorporate into the sequential program. This is another interesting area for further research.

### 2.3 Computational Results for PGRNET

The following test problems are organized in two main groups. Group 1 is a set of problems all having 2000 source nodes, 2000 destination nodes, and roughly 25000 arcs. We call these problems single-period problems because the system matrix does not have the multi-period structure shown in figure 1.2. The problems in Group 1 were generated in such a way that the granularity is somewhat different for each of the problems, and they are organized so that problem A has the largest granularity and problem G the smallest. An interesting observation is that the total run-time for the large-grained problems is considerably greater than the run-time for the small-grained problems. The sequential CPU time for problem A in Table 2.1 is 12503 seconds, while the sequential CPU time for problem G is only 208 seconds. This makes it clear that the large-grained problems are the most difficult ones, and they are the ones for which efficient parallel algorithms are most needed. All problems in Group 1 were run both with and without the variable list size heuristic. Table 2.1 gives run-time information with the heuristic, and Table 2.2 gives the same information without the heuristic. Group 2 is a set of problems which we generated in order to determine whether or not the efficiency of the parallel program depends on the ratio $(\#nodes)/(\#arcs)$ in a given problem. Each table gives results for a subgroup of problems with approximately the same granularity, but with differing numbers of arcs. The subgroups themselves are arranged in terms of decreasing granularity. All problems were run with and without the variable list size heuristic, and the results for these two runs are given in adjacent columns. If a problem name contains the number "32," then that problem was run with a fixed list size of 32. If a problem name contains the letters "va" then that problem was run using the variable list size heuristic. All of the problems in Group 2 have 1000 source nodes, and 1000 destination nodes. Problems having the number "06" in their name have between 8000 and 9000 arcs, and problems whose names include the numbers "10" or "20" have a number of arcs in the range 12000 to 13500 or 22000 to 23500 respectively. All problems in Groups 1 and 2 were generated by GTGEN [Chang and Engquist 86] which is based on NETGENG [Glover, et al., 78]. GTGEN allows the user to specify, roughly, the number of quasi-trees in the optimal basis.

For the Group 1 problems, figures 2.1 and 2.2 both indicate that the efficiency of PGRNET depends on the granularity of the problem being solved. The data on both of these tables shows that the speedup improves as the number of quasi-trees in the optimal basis is reduced from 1829 to 149. PGRNET solves problem C with a speedup of 8.8 when the fixed list size is used, and a speedup of 10.6 when the variable list size heuristic is used. Both of these results are superlinear because only 7 (the maximum number available at the time in the University of Wisconsin Balance 21000) processors were used to solve the problem. Problem C has the optimal granularity for PGRNET, because the speedup is lower for all problems with either larger or smaller granularity. Both runs of problem C yielded an optimal basis with 149 quasi-trees. When the number of quasi-trees in the basis is reduced below 149 the speedup of PGRNET seems to decrease substantially. For example, the speedup for problem A, with 60 quasi-trees, is only 5.5 when the variable list size heuristic is used, and only 4.8 when the fixed list size is used. Notice, however, that the speedup for problem A is superlinear when 4 processors are used. This happens because 4 processors can share 60 quasi-trees more easily than 7 processors. The speedup for problem G is sublinear. This is because PGRNET does more pivots and makes more candidate lists than GRNET, regardless of whether or not the list size is fixed. It appears from this that PGRNET does more work than GRNET to solve this small-grained problem. Note that the variable list size heuristic reduces the run time for problem G from 319 seconds to 208, an improvement of 34%, but the heuristic also reduces the speedup from 5.9 to 5.2. The reduction in speedup is probably due to the reduction in run-time in the sequential case, since the smaller sequential CPU time is more difficult to improve on. The results for problem F are similar. The speedup for problem F is sublinear, because PGRNET does more work than GRNET to solve this problem. The most interesting part of the results in Tables 2.1 and 2.2 are the superlinear speedups shown for problems C,D and

E. Looking at Table 2.2, the number of pivots needed by GRNET to solve problem C is 106559, while the number of pivots needed by PGRNET is only 85810. This helps to explain the superlinear speedup, because more CPU time is spent by GRNET executing pivots than by PGRNET. The number of candidate lists made by PGRNET, however, is 14396. This is more than twice the number of candidate lists made by GRNET, namely 6786. This suggests that more CPU time is spent by PGRNET computing reduced costs than by GRNET. Since making a candidate list might require computing the reduced cost on all of the arcs belonging to a given processor, making a candidate list can be more expensive than executing a pivot. Nevertheless, the superlinear speedups for problems C,D and E, indicate that the overhead paid by GRNET for doing more pivots outweighed the overhead paid by PGRNET for computing more reduced costs.

For the Group 2 problems, we tried to determine whether or not the efficiency of PGRNET depends on the ratio $(\#nodes)/(\#arcs)$ of the problem being solved. The results given in Tables 2.3, 2.4, 2.5, 2.6, 2.7 and 2.8 suggest that the performance of PGRNET improves as this ratio decreases. For most of the problems listed in these tables, the "20" problem has a greater speedup than the "06" problem. The "20" problem has a smaller ratio of nodes to arcs. Since a fairly wide range of problems is listed, this suggests that the efficiency of PGRNET improves as the ratio of nodes to arcs decreases. The other information in these tables seems to be consistent with the information in Tables 2.1 and 2.2. For example, the large-grained problems with the "va" suffix usually have better speedup than the corresponding "32" problems. Some exceptions to this are problems E-20 and D-06. The large-grain problems in Group 2 are similar to the large-grain problems in Group 1, in that the total number of pivots done by PGRNET is slightly less than the number done by GRNET, and the number of candidate lists made by PGRNET is considerably greater than the number made by GRNET. The extra pricing done by PGRNET explains the sublinear speedup obtained for Group 2, subgroup D. Another feature of the problems in Group 2 which is consistent with the problems in Group 1 is that there is an optimal granularity for PGRNET. The speedup is best for problems F and E, and it is worse for problems with either a larger or smaller granularity. The speedup for problem F-20-va is 8.9 on 7 processors, and the speedup is 7.4 for problem E-20-32. Both of these speedups are superlinear. For Group 1, the problems with optimal granularity had roughly 140 or 190 quasi-trees in the optimal basis, while the problems with optimal granularity in Group 2 have 80 to 140 quasi-trees in the optimal basis. This suggests that the performance of the program depends not just on the number of quasi-trees in the basis, but it must also depend on the number of quasi-trees relative to the total number of nodes in the problem. The effects of the variable list size heuristic are summarized in Tables 2.9 through 2.13. Each column gives the name of a problem, the CPU time required by GRNET to solve it using a fixed list size, the time required using the variable list size heuristic, and the percentage decrease (if any). The heuristic seems to give some improvement for almost all problems, but the most significant improvement is for the small-grained problems. The best improvement is for some small-grained multi-period problems which will be discussed in the next section.

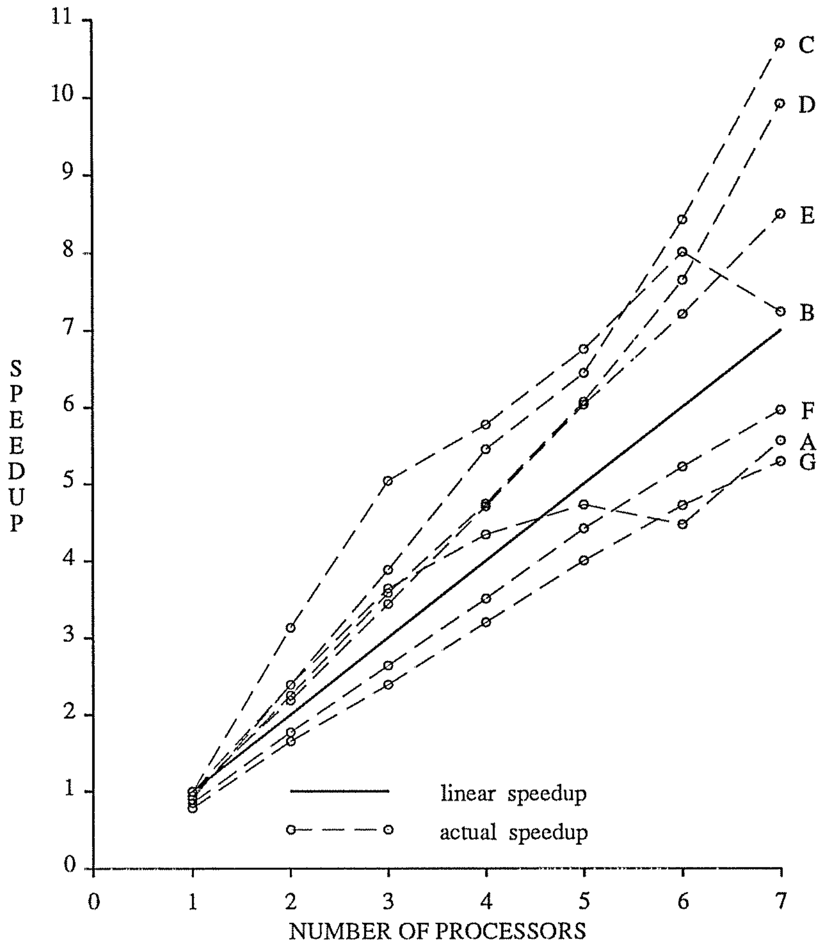

Figure 2.1 Speedup, Variable List Size Strategy

| **Variable List Size** | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
|---|---|---|---|---|---|---|---|
| No. qtrees at optimality | 60 | 81 | 149 | 194 | 287 | 1038 | 1829 |
| No. Nodes | 4000 | 4000 | 4000 | 4000 | 4000 | 4000 | 4000 |
| No. Arcs | 24621 | 24642 | 24710 | 24755 | 24848 | 25599 | 26390 |
| No. pivots sequential | 134351 | 125001 | 105778 | 96529 | 84237 | 52148 | 39424 |
| No. pivots 7 procs | 109737 | 102700 | 92339 | 88441 | 82363 | 51499 | 40422 |
| No. candidate lists sequential | 25492 | 25219 | 23185 | 23058 | 25050 | 52148 | 39424 |
| No. candidate lists 7 procs | 130951 | 79170 | 48700 | 47762 | 55625 | 50647 | 40504 |
| CPU secs sequential | 12503 | 9846 | 4590 | 3379 | 1920 | 348 | 208 |
| CPU secs 7 procs | 2248 | 1358 | 429 | 340 | 225 | 58 | 39 |
| speedup 7 procs | 5.5 | 7.2 | 10.6 | 9.9 | 8.50 | 5.9 | 5.2 |

Table 2.1 Group1, Variable List Size Strategy

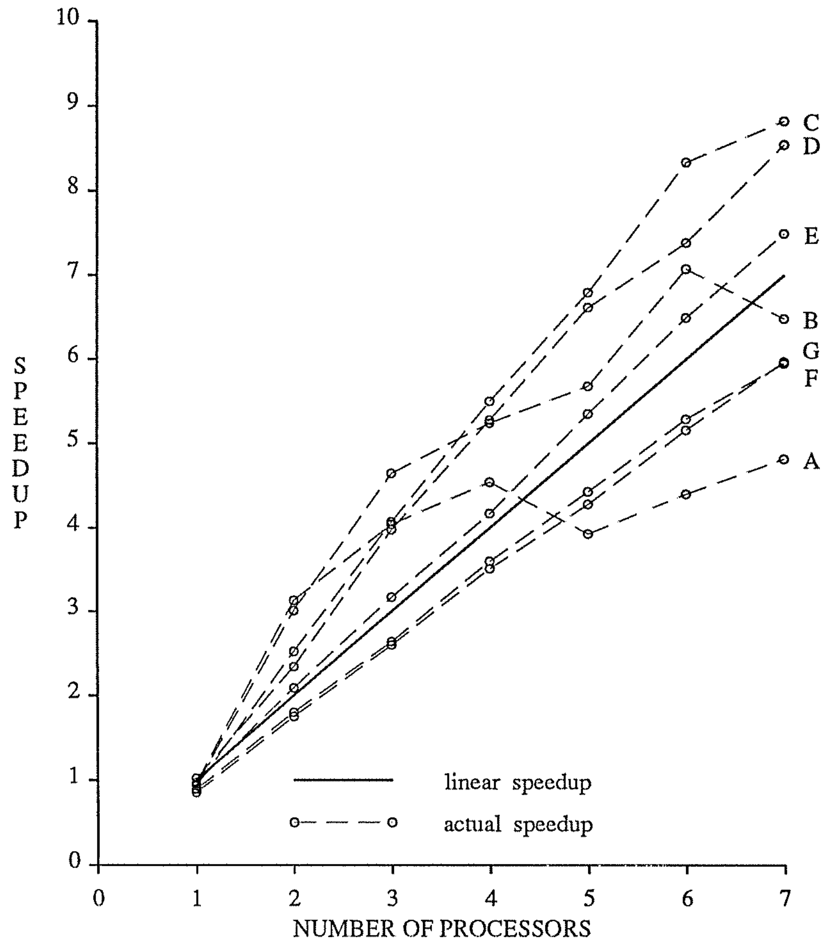

Figure 2.2 Speedup, Fixed List Size Strategy

| **List Size = 32** | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
|---|---|---|---|---|---|---|---|
| No. qtrees at optimality | 60 | 81 | 149 | 194 | 287 | 1038 | 1829 |
| No. Nodes | 4000 | 4000 | 4000 | 4000 | 4000 | 4000 | 4000 |
| No. Arcs | 24621 | 24642 | 24710 | 24755 | 24848 | 25599 | 26390 |
| No. pivots sequential | 132572 | 122776 | 106559 | 97191 | 82491 | 46593 | 35800 |
| No. pivots 7 procs | 109398 | 100790 | 85810 | 82057 | 72633 | 45668 | 35968 |
| No. candidate lists sequential | 8966 | 8085 | 6786 | 6148 | 5163 | 2912 | 2238 |
| No. candidate lists 7 procs | 126061 | 57282 | 14396 | 11015 | 7182 | 2980 | 2269 |
| CPU secs sequential | 12368 | 9356 | 4753 | 3663 | 1956 | 476 | 319 |
| CPU secs 7 procs | 2561 | 1442 | 538 | 428 | 261 | 80 | 53 |
| speedup 7 procs | 4.8 | 6.4 | 8.8 | 8.5 | 7.4 | 5.9 | 5.9 |

Table 2.2 Group 1, Fixed List Size Strategy

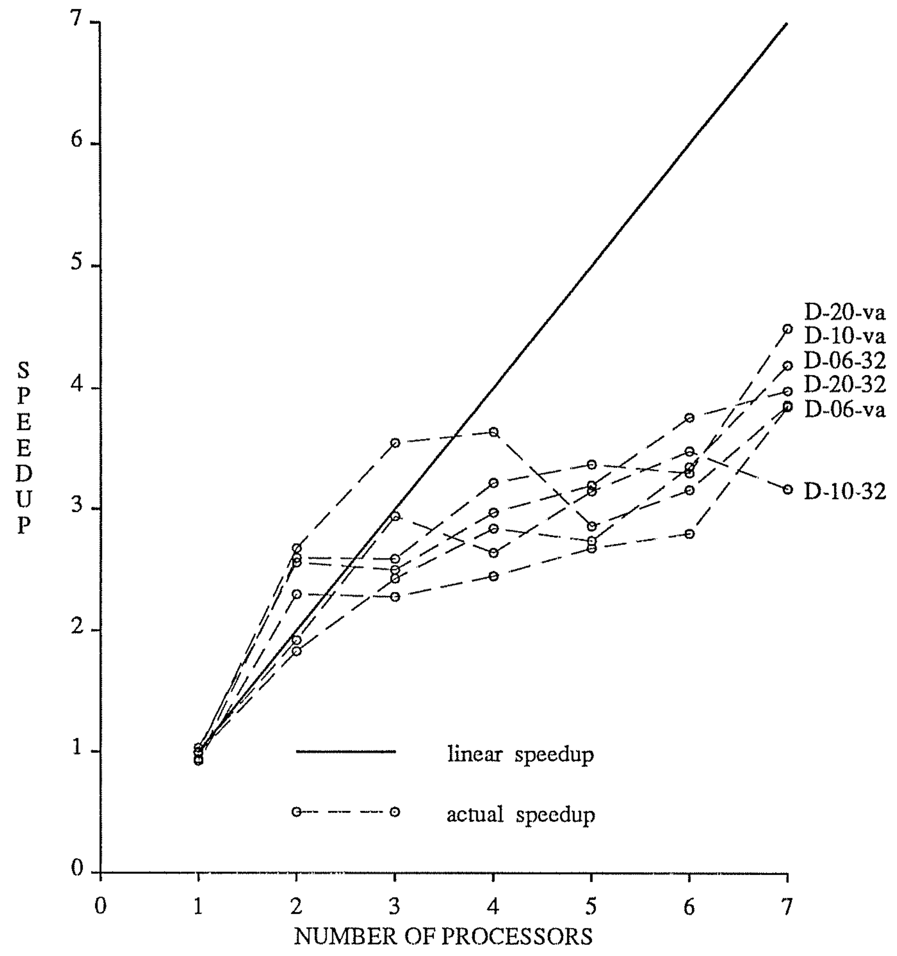

Figure 2.3 Speedup, Problem Set D

| List Size 32 / Var List Size | D-06-32 | D-06-va | D-10-32 | D-10-va | D-20-32 | D-20-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 48 | 48 | 42 | 42 | 44 | 44 |
| No. Nodes | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 |
| No. Arcs | 8381 | 8381 | 12356 | 12356 | 22309 | 22309 |
| No. pivots sequential | 32859 | 31888 | 55233 | 56233 | 111229 | 108264 |
| No. pivots 7 procs | 29093 | 29535 | 49730 | 48482 | 91318 | 90840 |
| No. candidate lists sequential | 2190 | 7462 | 3716 | 11550 | 7563 | 19640 |
| No. candidate lists 7 procs | 18614 | 38645 | 48790 | 57674 | 82934 | 77357 |
| CPU secs sequential | 1575 | 1338 | 3009 | 2989 | 6258 | 5855 |
| CPU secs 7 procs | 394 | 347 | 946 | 712 | 1617 | 1304 |
| speedup 7 procs | 3.9 | 3.8 | 3.1 | 4.1 | 3.8 | 4.4 |

Table 2.3 Group 2, Problem Set D

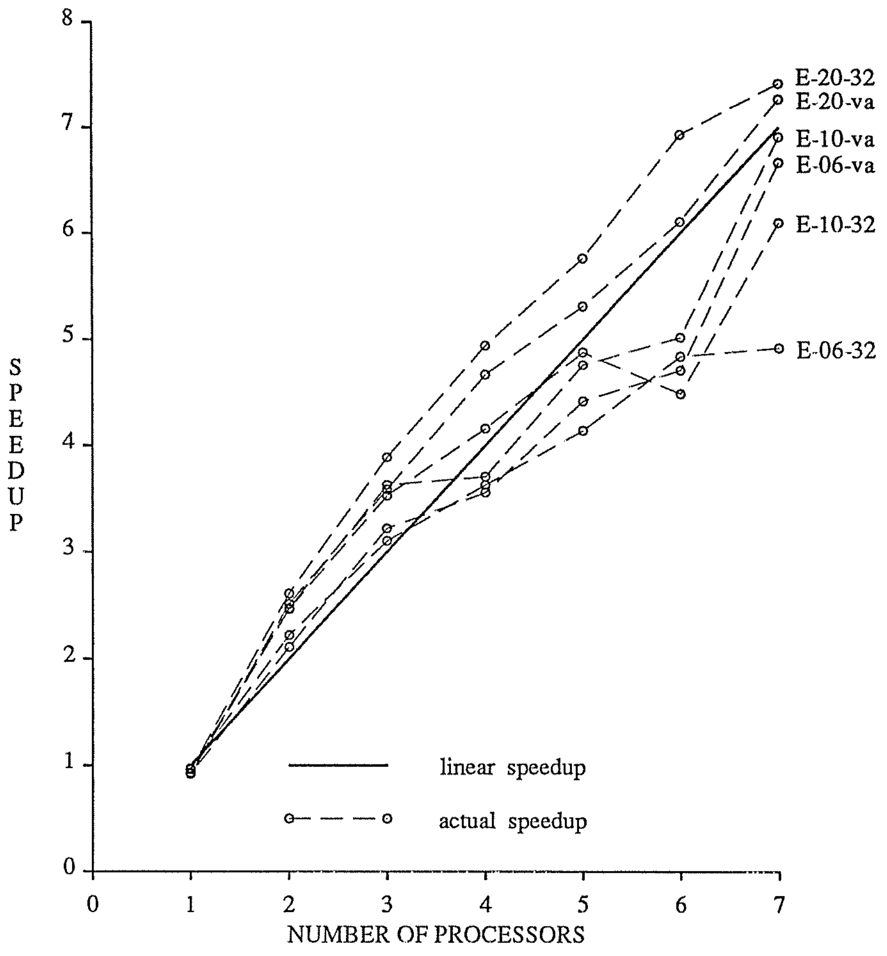

Figure 2.4 Speedup, Problem Set E

| List Size 32 / Var List Size | E-06-32 | E-06-va | E-10-32 | E-10-va | E-20-32 | E-20-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 80 | 80 | 67 | 67 | 76 | 76 |
| No. Nodes | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 |
| No. Arcs | 8413 | 8413 | 12381 | 12381 | 22341 | 22341 |
| No. pivots sequential | 28793 | 29465 | 50797 | 49853 | 97449 | 96526 |
| No. pivots 7 procs | 25903 | 27678 | 42542 | 44137 | 77599 | 81471 |
| No. candidate lists sequential | 1855 | 7991 | 3312 | 10767 | 6409 | 19236 |
| No. candidate lists 7 procs | 7573 | 23066 | 12055 | 27177 | 16800 | 43943 |
| CPU secs sequential | 953 | 900 | 2032 | 1958 | 3912 | 3794 |
| CPU secs 7 procs | 194 | 135 | 332 | 283 | 527 | 521 |
| speedup 7 procs | 4.9 | 6.6 | 6.1 | 6.9 | 7.4 | 7.2 |

Table 2.4 Group 2, Problem Set E

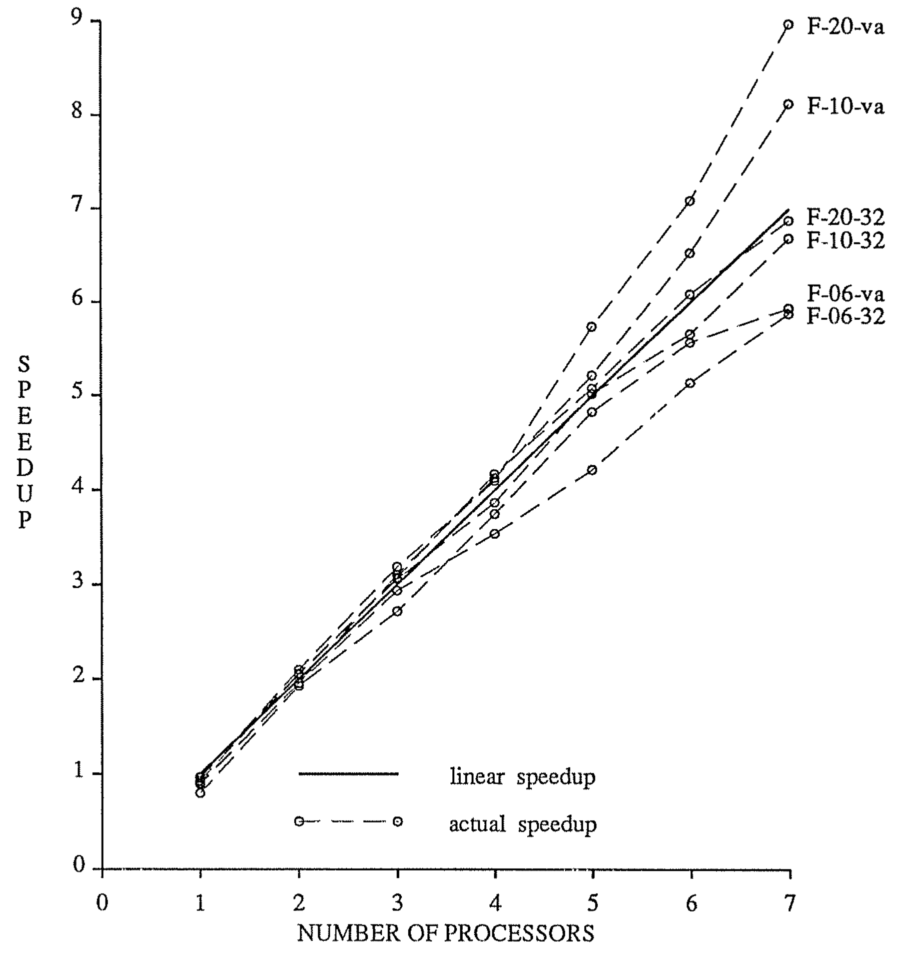

Figure 2.5 Speedup, Problem Set F

| List Size 32 / Var List Size | F-06-32 | F-06-va | F-10-32 | F-10-va | F-20-32 | F-20-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 139 | 139 | 144 | 144 | 147 | 147 |
| No. Nodes | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 |
| No. Arcs | 8472 | 8472 | 12458 | 12458 | 22412 | 22412 |
| No. pivots sequential | 23865 | 24668 | 39676 | 41035 | 77510 | 78628 |
| No. pivots 7 procs | 21702 | 24625 | 34591 | 37639 | 66856 | 72495 |
| No. candidate lists sequential | 1498 | 8097 | 2495 | 11190 | 4899 | 17890 |
| No. candidate lists 7 procs | 2635 | 17154 | 3981 | 22060 | 8196 | 42201 |
| CPU secs sequential | 482 | 416 | 840 | 810 | 1780 | 1768 |
| CPU secs 7 procs | 81 | 70 | 125 | 99 | 258 | 196 |
| speedup 7 procs | 5.8 | 5.9 | 6.6 | 8.1 | 6.8 | 8.9 |

Table 2.5 Group 2, Problem Set F

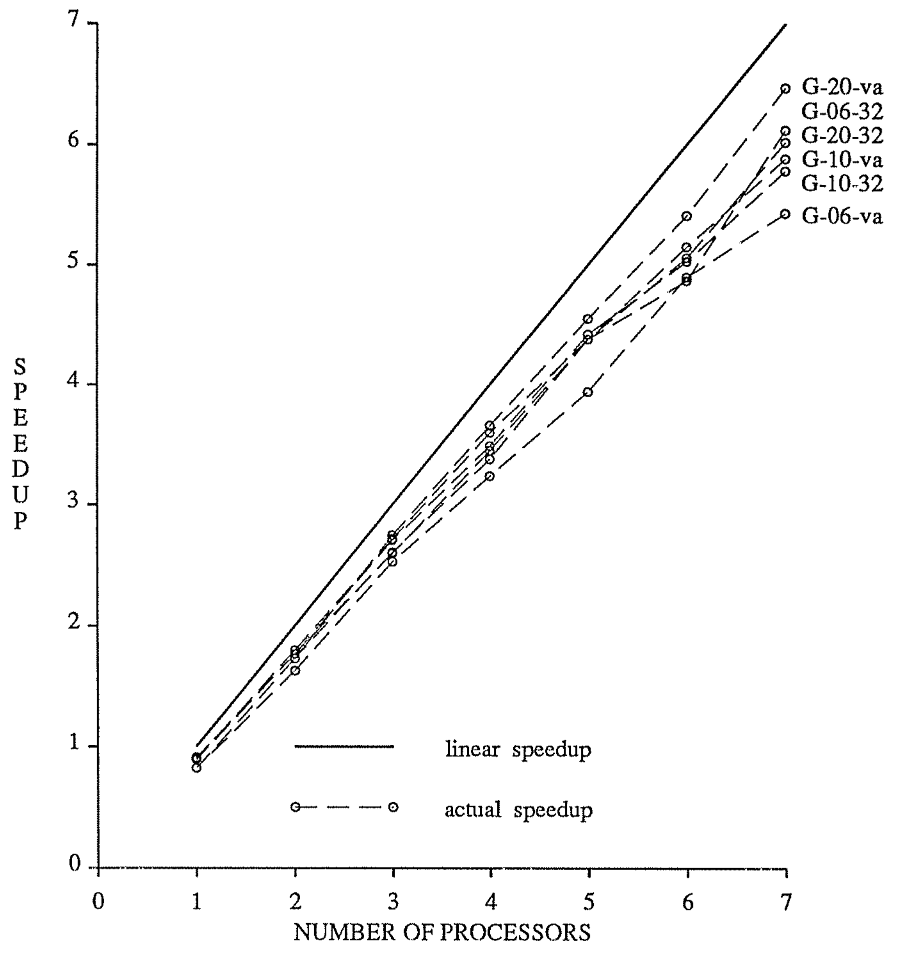

Figure 2.6 Speedup, Problem Set G

| **List Size 32 / Var List Size** | **G-06-32** | **G-06-va** | **G-10-32** | **G-10-va** | **G-20-32** | **G-20-va** |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 333 | 333 | 332 | 332 | 323 | 323 |
| No. Nodes | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 |
| No. Arcs | 8666 | 8666 | 12646 | 12646 | 22588 | 22588 |
| No. pivots sequential | 17516 | 19194 | 28130 | 30907 | 54407 | 58453 |
| No. pivots 7 procs | 16524 | 19725 | 27150 | 31732 | 51182 | 59151 |
| No. candidate lists sequential | 1095 | 13036 | 1759 | 17837 | 3401 | 23813 |
| No. candidate lists 7 procs | 1210 | 19508 | 2077 | 28776 | 4035 | 50088 |
| CPU secs sequential | 202 | 150 | 336 | 267 | 684 | 568 |
| CPU secs 7 procs | 33 | 27 | 58 | 45 | 113 | 87 |
| speedup 7 procs | 6.1 | 5.4 | 5.7 | 5.8 | 6.0 | 6.4 |

Table 2.6 Group 2, Problem Set G

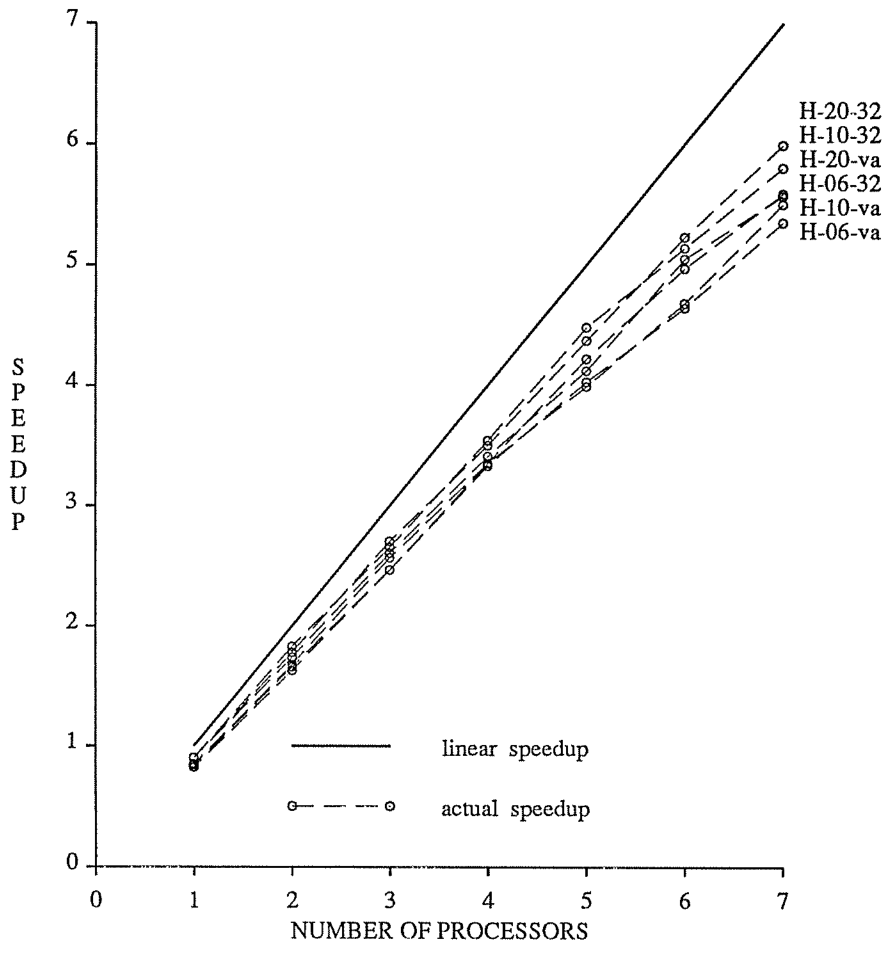

Figure 2.7 Speedup, Problem Set H

| List Size 32 / Var List Size | H-06-32 | H-06-va | H-10-32 | H-10-va | H-20-32 | H-20-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 538 | 538 | 513 | 513 | 540 | 540 |
| No. Nodes | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 |
| No. Arcs | 8871 | 8871 | 12827 | 12827 | 22805 | 22805 |
| No. pivots sequential | 14539 | 16018 | 23674 | 26047 | 42756 | 47205 |
| No. pivots 7 procs | 14703 | 16390 | 23193 | 26528 | 41999 | 48602 |
| No. candidate lists sequential | 910 | 11984 | 1481 | 17538 | 2674 | 31736 |
| No. candidate lists 7 procs | 982 | 16652 | 1571 | 27047 | 2827 | 46468 |
| CPU secs sequential | 140 | 96 | 238 | 167 | 432 | 308 |
| CPU secs 7 procs | 25 | 18 | 41 | 30 | 72 | 55 |
| speedup 7 procs | 5.5 | 5.3 | 5.8 | 5.5 | 5.9 | 5.5 |

Table 2.7 Group 2, Problem Set H

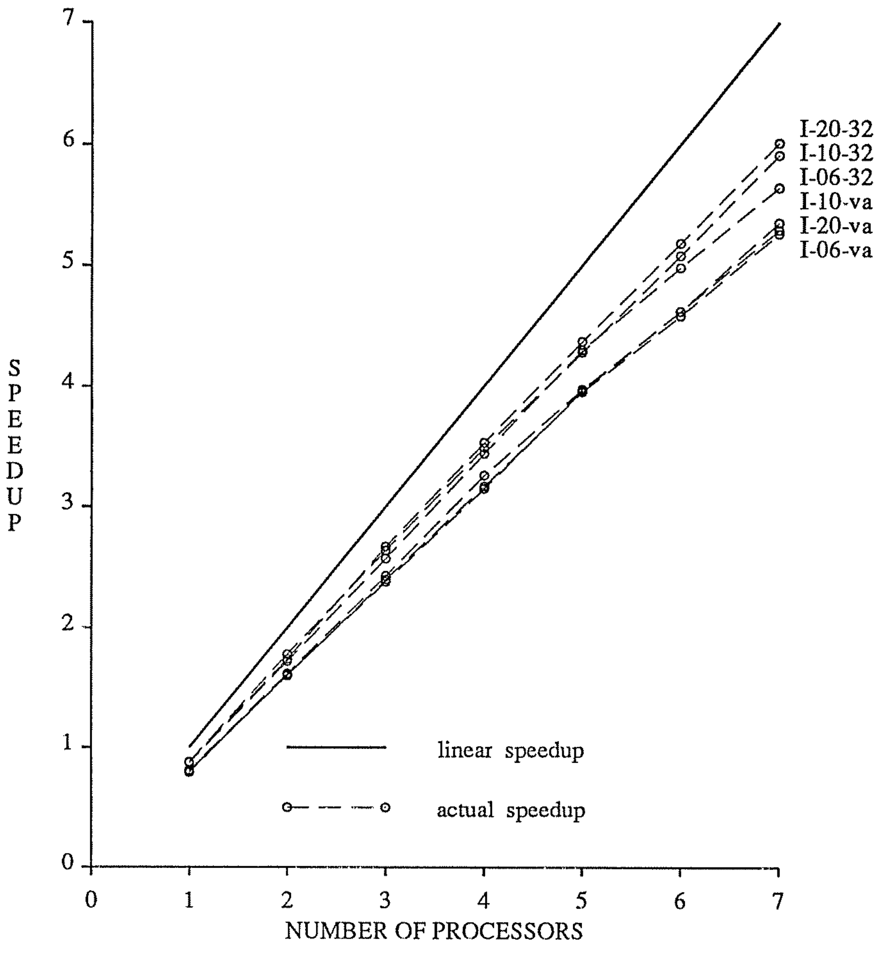

Figure 2.8 Speedup, Problem Set I

| List Size 32 / Var List Size | I-06-32 | I-06-va | I-10-32 | I-10-va | I-20-32 | I-20-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 921 | 920 | 912 | 912 | 914 | 914 |
| No. Nodes | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 |
| No. Arcs | 9254 | 9254 | 13226 | 13226 | 23179 | 23179 |
| No. pivots sequential | 12112 | 13426 | 18114 | 20171 | 33181 | 36569 |
| No. pivots 7 procs | 12155 | 13247 | 18110 | 20243 | 32767 | 37147 |
| No. candidate lists sequential | 757 | 13426 | 1133 | 20171 | 2075 | 36569 |
| No. candidate lists 7 procs | 781 | 13307 | 1154 | 20337 | 2084 | 37316 |
| CPU secs sequential | 103 | 66 | 158 | 101 | 291 | 185 |
| CPU secs 7 procs | 18 | 12 | 26 | 19 | 48 | 35 |
| speedup 7 procs | 5.6 | 5.2 | 5.9 | 5.3 | 6.0 | 5.2 |

Table 2.8 Group 2, Problem Set I

| Problem Name | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| CPU time with list size 32 | 12368 | 9356 | 4753 | 3663 | 1956 | 476 | 319 |
| CPU time with var. list size | 12503 | 9846 | 4590 | 3379 | 1920 | 348 | 208 |
| Percent decrease in CPU time | 1% | ---- | 3% | 7% | 1% | 26% | 34% |

Table 2.9 Group 1, Improvement of Sequential CPU Time

| Problem Name | D-06 | D-10 | D-20 | E-06 | E-10 | E-20 |
|---|---|---|---|---|---|---|
| CPU time with list size 32 | 1575 | 3009 | 6258 | 953 | 2032 | 3912 |
| CPU time with var. list size | 1338 | 2989 | 5855 | 900 | 1958 | 3794 |
| Percent decrease in CPU time | 15% | 1% | 6% | 5% | 3% | 3% |

Table 2.10 Group 2, Improvement of Sequential CPU Time, subgroups D and E

| Problem Name | F-06 | F-10 | F-20 | G-06 | G-10 | G-20 |
|---|---|---|---|---|---|---|
| CPU time with list size 32 | 482 | 840 | 1780 | 202 | 336 | 684 |
| CPU time with var. list size | 416 | 810 | 1768 | 150 | 267 | 568 |
| Percent decrease in CPU time | 13% | 3% | 1% | 25% | 20% | 16% |

Table 2.11 Group 2, Improvement of Sequential CPU Time, subgroups F and G

| Problem Name | H-06 | H-10 | H-20 | I-06 | I-10 | I-20 |
|---|---|---|---|---|---|---|
| CPU time with list size 32 | 140 | 238 | 432 | 103 | 158 | 291 |
| CPU time with var. list size | 96 | 167 | 308 | 66 | 101 | 185 |
| Percent decrease in CPU time | 31% | 29% | 28% | 35% | 36% | 36% |

Table 2.12 Group 2, Improvement of Sequential CPU Time, subgroups H and I

# 3. A Variant of PGRNET for Multiperiod Problems

## 3.1 Multiperiod problems

Multiperiod problems have the following form:

$$\min \quad cx$$

$$\text{s.t.} \quad Gx = b$$

$$0 \leq x \leq u$$

As before, $G$ has no more than two non-zero elements in each column, and in addition, $G$ has the form given in figure 1.2. In our test problems, there are two to twelve blocks of arcs running diagonally through the matrix $G$, and a small number of variables extending between adjacent blocks. In applications, the blocks correspond to different time periods, and the arcs between blocks represent inventory or back order (or backlogging) arcs. In-depth studies of multi-period problems can be found in [Fong and Srinivasan 76], [Hausman and Gilmour 67], and [Klingman and Mote 82]. MPGEN [Chang, et al., 87] generates such problems, and generates backlogging arcs with a user specified density. MPGEN is based on GTGEN [Chang and Engquist 86].

Since [Chang, et al., 87] reported quite good results for multi-period problems using a code named PAGEN-K on the CRYSTAL Multicomputer [Chang, et al., 87], we decided to use a similar approach to improve PGRNET. We will briefly describe PAGEN-K before discussing our algorithm, MPGRNET, in detail. PAGEN-K exploits the fact that the majority of all pivots are done on arcs joining nodes in the same period (block). PAGEN-K uses $(k+1)$ processors for a $k$ period problem, and it gives all of the nodes corresponding to one period to a single processor. A processor $p$ then solves its local problem to optimality, ignoring the cross arcs that extend to processors $p-1$ and $p+1$. When all processors have finished their local problems, a master processor directs a transfer of quasi-trees between processors, giving to each processor a new local problem. Initially the cross arcs are the backlogging arcs, but after the first transfer of quasi-trees, the set of cross arcs can include arcs that are local to a period. This approach yielded some impressive speed-ups on CRYSTAL provided that communication time was neglected.

In our algorithm, MPGRNET, each processor is given a list of arcs rather than a list of nodes. Each list is made up of arcs connecting nodes in a single-period or in a group of periods allocated to a given processor, and no backlogging arcs appear in any of the lists. During STAGE 0, processors chose pivot eligible arcs from their lists and perform pivots without the need for locking quasi-trees. Locking quasi-trees is an expensive operation which is unnecessary as long as backlogging arcs are not used as pivot arcs. Much of the efficiency of MPGRNET is due to the elimination of quasi-tree locking. Since processors pivot only on local arcs during STAGE 0, this stage is similar to PAGEN-K. When a processor $p$ completes STAGE 0, it goes to STAGE 1 and waits for the other processors to get there. This is the only part of the algorithm in which processors are idle. If one processor needed much more time to complete its local optimization than the others, then a significant loss in efficiency might result from processors sitting idle in STAGE 1. In our test problems, however, processors solved their local problems in nearly the same amount of time, and therefore no processor wasted much time waiting in STAGE 1. STAGE 2 and STAGE 3 are the same as STAGE 1 and STAGE 2 in PGRNET. In other words, in STAGE 2 of MPGRNET, backlogging arcs are added to the lists, and processors perform pivots simultaneously. Flagging quasi-trees is necessary in STAGE 2 because now it may happen that two or more processors will need to modify the same tree. In STAGE 3, optimality is achieved and verified.

In summary, the (parallel) MPGRNET algorithm is as follows:

INITIALIZATION

Processor 1 generates an initial feasible solution and makes a list of local arcs for each of the processors. Processor 1 then requests additional processors and all processors go to Stage 0.

STAGE 0 (*parallel pivoting without locking*)

All processors develop candidate lists and perform pivots without locking quasi-trees. Processors remain in Stage 0 until a candidate list is made with 4 or fewer entries. When this happens, processors proceed to Stage 1.

STAGE 1

Processors wait here until all processors get to this point, and then all proceed to Stage 2.

STAGE 2 (*parallel pivoting with locking*)

Same as STAGE 1 of PGRNET

STAGE 3 (*verification of optimality*)

Same as STAGE 2 of PGRNET

MPGRNET has the same data structures as PAGEN-K for storing tree functions and problem data. MPGRNET has an additional boolean array of length $m$ which contains the value (locked or unlocked) of each quasi-tree lock. Thus, MPGRNET has memory demands that are slightly greater than those of PAGEN-K. The algorithmic differences between the two programs are more pronounced. MPGRNET executes STAGE 0 only once. During this stage, processors solve their local problems and use this solution as an advanced start for STAGE 2. Mutual exclusion from shared data structures is provided during STAGE 2 by hardware locks, and optimality is verified in parallel. (In PAGEN-K, processors solve a sequence of local problems given to them by the master processor. Mutual exclusion during most of the run is provided by assigning quasi-trees to processors. At the end of the PAGEN-K run, all quasi-trees are sent to one processor which executes the remaining pivots and verifies optimality.)

## 3.2 Computational Results for Multiperiod Problems

The following test problems are organized into Groups 3,4 and 5. Group 3 is a set of multi-period problems generated by MPGEN [Chang, et al., 87]. Each table gives results for a subgroup of problems with roughly the same granularity, but with differing densities in the backlogging arcs. All of the problems in Group 3 are multi-period problems with 12 periods, and each period has 100 source nodes and 400 destination nodes. The names for these problems have three parts. The left part is a 2-digit number indicating the granularity of the periods. If this number is small, the number of quasi-trees in the optimal basis is small, and thus, the granularity of the periods is large. The middle part of a name is a 1-digit number indicating the density of the backlogging arcs. If this number is 1, roughly 2% of all arcs are backlogging arcs. If this number is 5 or 9, roughly 4% or 7% of all arcs are backlogging arcs respectively. The last part of each name is a "32" or a "va." The number "32" indicates that the problem was run using the fixed list size strategy, and the letters "va" indicate that the variable list size strategy was used. The data for the "va" and "32" runs of any given problem are given in adjacent columns of Tables 3.1 through 3.5. Group 4 is a selection of problems from Group 3 solved by PGRNET rather than MPGRNET. The results from Group 4 are useful in determining the decrease in CPU time that results from the advanced start heuristic used by MPGRNET in STAGE 0. These results are given in Tables 3.6 and 3.7. Group 5 is a set of single-period problems solved by PGRNET. Table 3.8 gives results for these problems when the list size is fixed at 32, and Table 3.9 gives results for the variable list size heuristic. These tables fit best with the multi-period results because they have the same number of source nodes and destination nodes as the multi-period problems, and they have roughly the same number of arcs as the multi-period problems. The problems in Group 5 help to indicate how the presence of backlogging arcs in the problem affect the performance of PGRNET.

For the Group 3 problems, the specialization of PGRNET, called MPGRNET, was run on a Sequent Balance 21000 at Argonne National Laboratories. This machine has 24 processors, and a single user may use up to 23 of these in executing a program. We used at most 12 processors.

MPGRNET yielded superlinear speedup for all of the problems, as shown in Table 3.1, and the superlinearity of the speedup for most of the problems improves as the number of processors increases. For example, $speedup(6)/6 = 1.9$ and $speedup(12)/12 = 3.0$ for problem 05-9-va. $Speedup(6)/6 = 2.2$, and $speedup(12)/12 = 2.8$ for problem 05-9-32. The improvement in this ratio and the superlinear speedup both suggest that the solution technique used by the parallel program is superior to the technique used by the sequential program. The sequential program, GRNET, could probably be improved by incorporating into it the advanced start used by MPGRNET. GRNET could solve subproblems corresponding to different periods before solving the problem as a whole. This would mimic STAGE 0, the stage in MPGRNET in which processors solve local problems in parallel.

The problems listed in tables 3.1 and 3.2 are all large-grained problems. For most of these, the speedup for the "va" problem is greater than the speedup for the "32" problem. More testing is needed to see if this continues when the grain size is made even larger. For problems with smaller granularity, the "32" problems show a better speedup. As with the single-period problems, the variable list size heuristic seems to yield a decrease in the run-time for almost all multi-period problems. Sequential CPU times for multi-period problems are given in Tables 3.10,3.11 and 3.12. The times for both the fixed list size strategy and the variable list size strategy are given, along with the percent improvement. The best improvement was 45% for problem 90-1. The best improvement for any single-period problem was 36% . Overall, the variable list size heuristic seems to have a more pronounced effect for the multi-period problems than it does for the single-period problems.

A noticeable feature of the problems in Tables 3.1 through 3.3 is that the speedup improves as the density of the backlogging arcs increases. For example, maximum speedup for problem 05-1-va was 14.9, while the maximum speedup for problem 05-9-va is 36.9. The maximum speedup for problem 10-1-32 is 14.5, while the maximum speedup for problem 10-9-32 is 28.9. This is a result of the fact that increasing the density of the backlogging arcs makes the problem more difficult for the sequential code, but has little effect on the performance of the parallel code. Notice also that the number of quasi-trees in the optimal basis is the same for all of the problems in any given subgroup. This means that the backlogging arcs probably don't have much effect on the nature of the optimal solution. They do, however, increase the total run time by a substantial amount. The sequential CPU time for problem 05-32-1 is 1009 seconds, while the sequential CPU time for 05-32-9 is 3072 seconds, an increase of 200% .

To explain the superlinear speedup in terms of the amount of work done by the sequential program and the parallel program, one can look at the number of pivots executed and the number of candidate lists made. GRNET does 44046 pivots to solve problem 05-9-32, while MPGRNET solves it with only 33452 pivots. Clearly GRNET is spending more CPU time executing pivots. The number of candidate lists made by the programs gives an indication of the amount of work done computing reduced costs. GRNET made 3088 lists while solving problem problem 05-9-32, while MPGRNET made 2930. Apparently both programs did roughly the same amount of work in making candidate lists, so the total speedup of 34.4 for this problem must be explained by the discrepancy in the number of pivots. But if this were the only factor, then the speedup would be roughly 16, not 34.4. It is possible that due to the locking of quasi-trees in STAGE 2 and STAGE 3, MPGRNET is more likely than the sequential program to pivot on arcs that connect leaves of a single quasi-tree, as opposed to arcs that connect two quasi-trees. If this were true, then MPGRNET might develop a basis forest in which the nodes are more evenly distributed between the quasi-trees. The effect of this would be to make pivots less costly for MPGRNET than for GRNET. This is one possible explanation for the high speedup of problem 05-9-32.

The speedup results listed in Table 3.4 are all slightly sublinear. Looking at the "32" problems, one sees that the sequential program and the parallel program both do roughly the same number of pivots, and they both make roughly the same number of candidate lists. It is difficult to judge how much work is done by the two programs making candidate lists for the "va" problems because the sequential program and the parallel program can select different list sizes. GRNET solved the "va" problems with slightly more pivots than MPGRNET, but somehow MPGRNET did more work than GRNET to solve the problems in this subgroup because the speedup is sublinear. The speedup for the "va" problems is less than 11 for all of these problems, and the speedup for all of the "32" problems is greater than 11. So in this respect, the fixed list size strategy seems to be behaving better than the variable list size strategy for these problems. The variable list size strategy, however, yields a lower overall run time. The speedup results listed in Table 3.5 are similar to those of Table 3.4 in the sense that the speedups are higher for the "32" problems than for the "va" problems. For this subgroup, the variable list size heuristic makes quite an improvement in total run-time over the fixed list size strategy. GRNET solved problem 90-1-32 in 210 seconds, and it solved problem 90-1-va in 115 seconds This means that the variable list size yielded an improvement of 45% .

Tables 3.6 and 3.7 show the results for the problems in Group 4. These are multi-period problems solved by PGRNET as if they were single-period problems. The problems listed in Tables 3.6 and 3.7 are the same as the problems listed in Tables 3.1 and 3.5 respectively. The speedups listed on Table 3.6 are all superlinear. This means that problems 05-1, 05-5 and 05-9 have a granularity that permits efficient solution by PGRNET, and this helps to explain the highly superlinear results given in Table 3.1. Since MPGRNET uses PGRNET in STAGE 2 and STAGE 3, the high performance of PGRNET for this subgroup gives MPGRNET an extra boost.

Comparing Tables 3.1 and 3.6, the parallel CPU time for problem 05-9-va is 136 seconds when solved by PGRNET and only 75 seconds when solved by MPGRNET, a 44% improvement. MPGRNET solves (in parallel) subproblems corresponding different periods before solving the problem as a whole. The reduction in CPU time from 136 seconds to 75 seconds shows that the advanced start provided by the solution of these subproblems is a heuristic which gives a substantial improvement in overall runtime. This heuristic gives a slight improvement for problem 05-1, a larger improvement for problem for problem 05-5, and the the largest improvement for problem 05-9. Since the sequential CPU time for problem 05-9 is almost three times as great as the sequential CPU time for problem 05-1, the heuristic seems to be working the best for the problems which are most difficult in the sense that they require much CPU time.

Comparing Tables 3.5 and 3.7, the parallel CPU time for problem 90-9-32 is 18 seconds when solved by PGRNET and 15 seconds when solved by MPGRNET. If the same comparison is made for problems 90-1 and 90-5, the CPU time reduction is similar or smaller. This means that for these small-grained multi-period problems, the advanced start heuristic gives only a slight improvement.

Data for the Group 5 problems given in Tables 3.8 and 3.9 give some useful information about the performance of PGRNET when solving single-period problems similar in topology to the multi-period problems. The problems in Tables 3.8 and 3.9 have the same ratio of source nodes to destination nodes and roughly same number of arcs as the multi-period problems. Also, the number of quasi-trees in the optimal bases of these problems ranges from 85 to 1097, and the number of quasi-trees in the optimal bases of the multi-period problems ranges from 84 to 1104. So in these respects, the problems in Group 5 are similar to the multi-period problems. The speedups, however, are not as impressive as the speedups given in Tables 3.6 and 3.7. In particular, the speedup for problem "a" run on 7 processors is 8.7 with variable list size and 7.1 with fixed list size. Some of the multi-period problems listed in Table 3.6 show a much higher speedup (ranging from 8 to 15) on 7 processors, even though they have nearly the same number of quasi-trees in the optimal basis. This indicates that the block structure of the multi-period problems makes them more amenable to parallel solution. This result suggests that the solution of single-period problems may benefit from a warm start in which the optimal quasi-tree structure of a related problem is used to establish an initial arc allocation to processors.

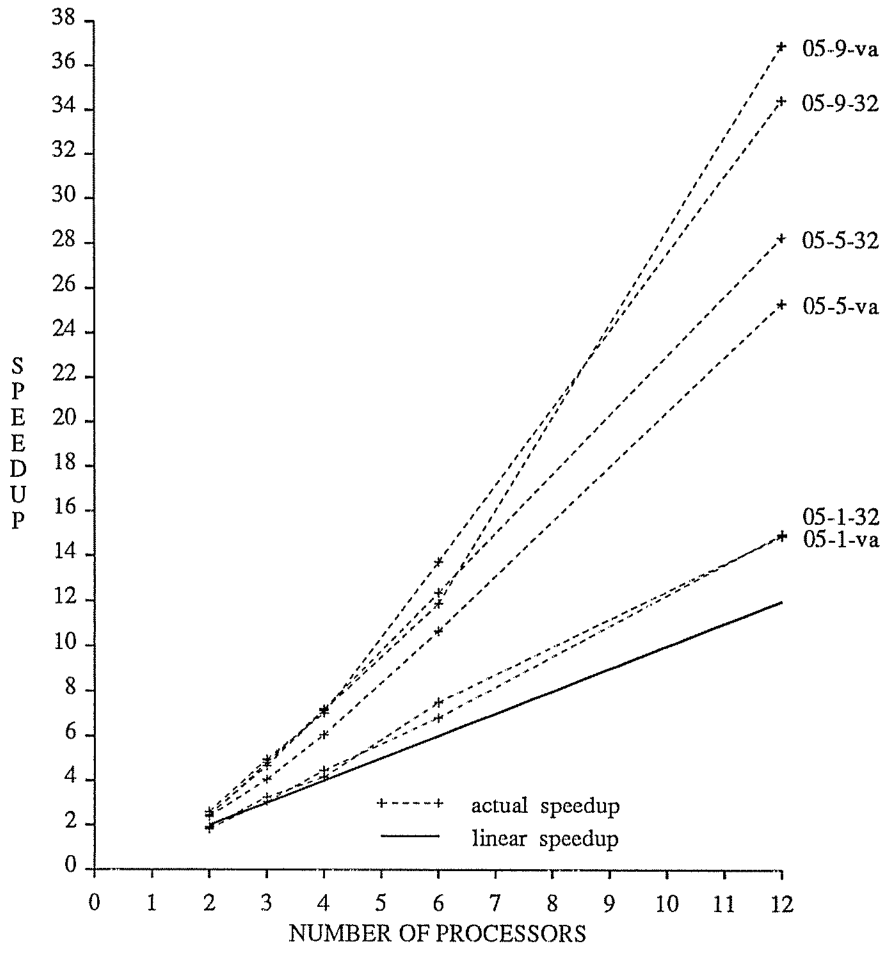

Figure 3.1 Speedup Graph for Problem Set 05

| list size = 32 / var. list size | 05-1-32 | 05-1-va | 05-5-32 | 05-5-va | 05-9-32 | 05-9-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 84 | 84 | 84 | 84 | 84 | 84 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 14952 | 14952 | 15367 | 15367 | 15742 | 15742 |
| No. pivots sequential<br>No. pivots 12 procs | 32523<br>31578 | 33374<br>32345 | 38405<br>32441 | 38295<br>33400 | 44046<br>33452 | 42676<br>34440 |
| No. cand. lists sequential<br>No. cand. lists 12 procs | 2208<br>2189 | 13294<br>7052 | 2696<br>2513 | 13666<br>9380 | 3088<br>2930 | 12476<br>10294 |
| CPU secs sequential<br>CPU secs 12 procs | 1009<br>67 | 964<br>64 | 2157<br>76 | 1848<br>72 | 3072<br>89 | 2776<br>75 |
| speedup 12 procs | 14.9 | 14.9 | 28.3 | 25.3 | 34.4 | 36.9 |

Table 3.1 Multi-Period Problem Set 05

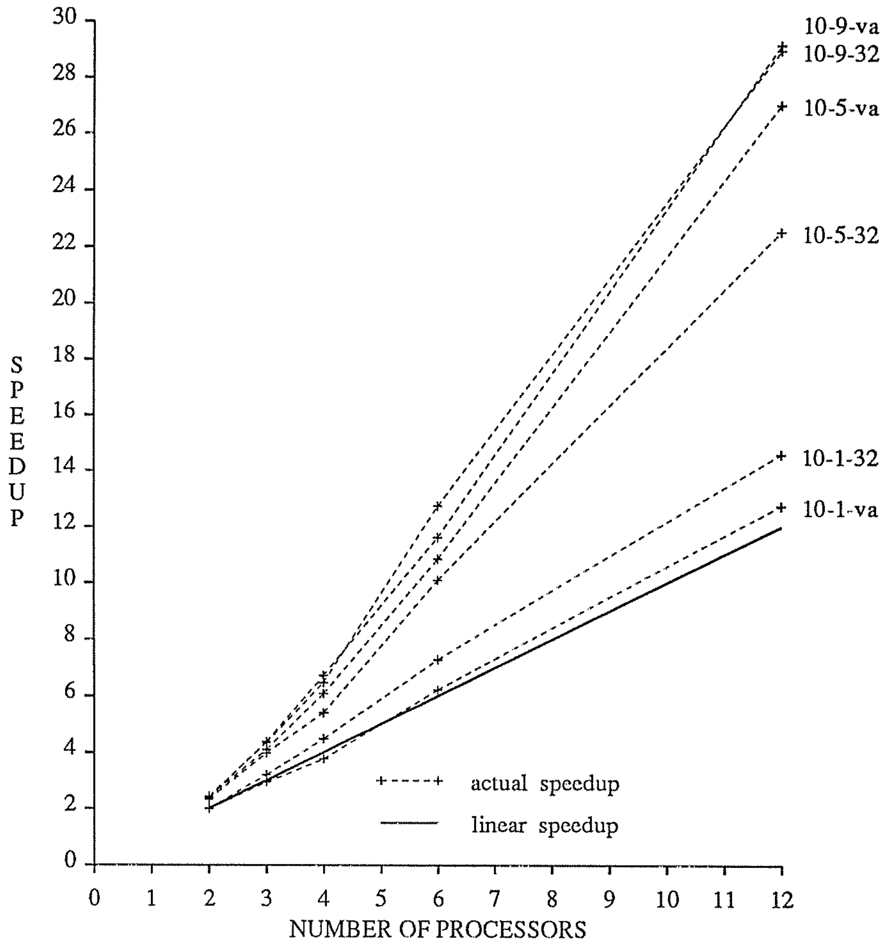

Figure 3.2 Speedup Graph for Problem Set 10

| list size = 32 / var. list size | 10-1-32 | 10-1-va | 10-5-32 | 10-5-va | 10-9-32 | 10-9-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 132 | 132 | 132 | 132 | 132 | 132 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 15037 | 15037 | 15433 | 15433 | 15793 | 15793 |
| No. pivots sequential<br>No. pivots 12 procs | 30656<br>29828 | 32023<br>30807 | 35302<br>30413 | 35730<br>31016 | 39462<br>31560 | 39157<br>32214 |
| No. cand. lists sequential<br>No. cand. lists 12 procs | 2054<br>1994 | 13341<br>6901 | 2395<br>2184 | 13968<br>9176 | 2716<br>2201 | 12996<br>10162 |
| CPU secs sequential<br>CPU secs 12 procs | 779<br>53 | 702<br>55 | 1368<br>60 | 1440<br>53 | 1929<br>66 | 1899<br>65 |
| speedup 12 procs | 14.5 | 12.7 | 22.5 | 27.0 | 28.9 | 29.1 |

Table 3.2 Multi-Period Problem Set 10

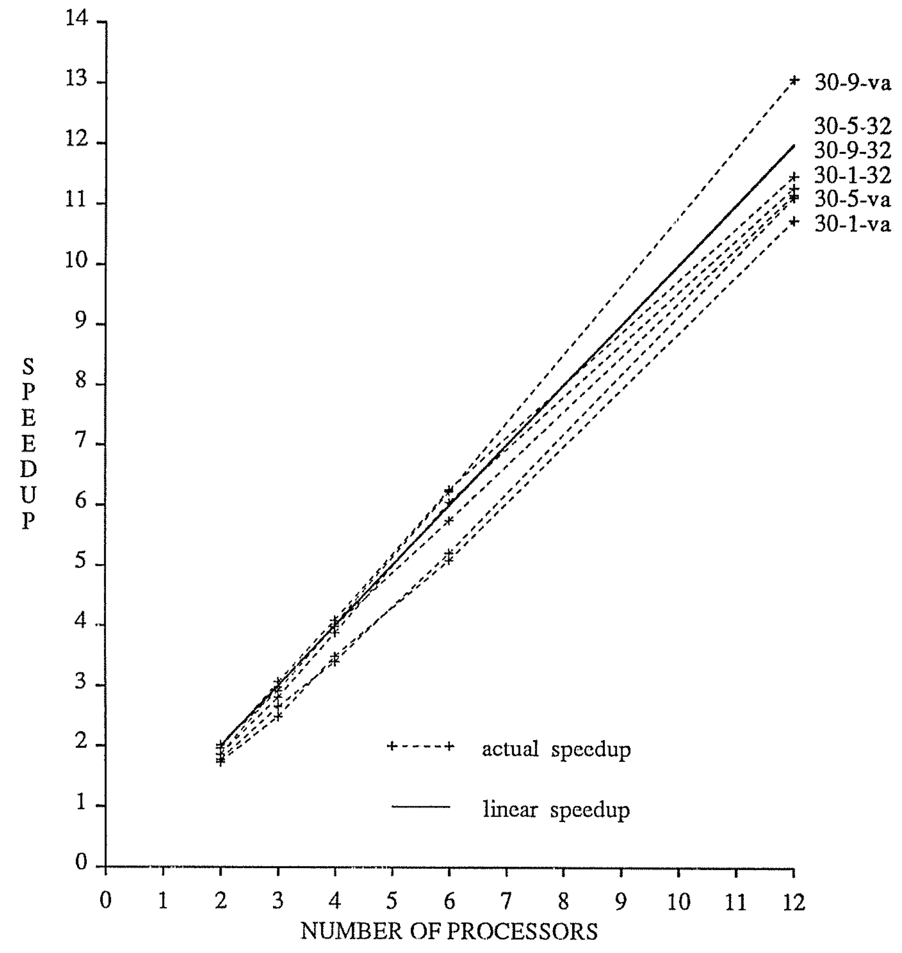

Figure 3.3 Speedup Graph for Problem Set 30

| **list size = 32 / var. list size** | **30-1-32** | **30-1-va** | **30-5-32** | **30-5-va** | **30-9-32** | **30-9-va** |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 408 | 408 | 408 | 408 | 408 | 408 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 15528 | 15528 | 15817 | 15817 | 16087 | 16087 |
| No. pivots sequential | 25108 | 27311 | 26258 | 27599 | 27072 | 28645 |
| No. pivots 12 procs | 24340 | 25377 | 25049 | 25789 | 25574 | 26194 |
| No. cand. lists sequential | 1597 | 16335 | 1668 | 14598 | 1719 | 14219 |
| No. cand. lists 12 procs | 1585 | 10397 | 1626 | 10606 | 1702 | 10826 |
| CPU secs sequential | 303 | 232 | 356 | 270 | 373 | 340 |
| CPU secs 12 procs | 27 | 21 | 31 | 24 | 33 | 26 |
| speedup 12 procs | 11.1 | 10.7 | 11.48 | 11.1 | 11.2 | 13.0 |

Table 3.3 Multi-Period Problem Set 30

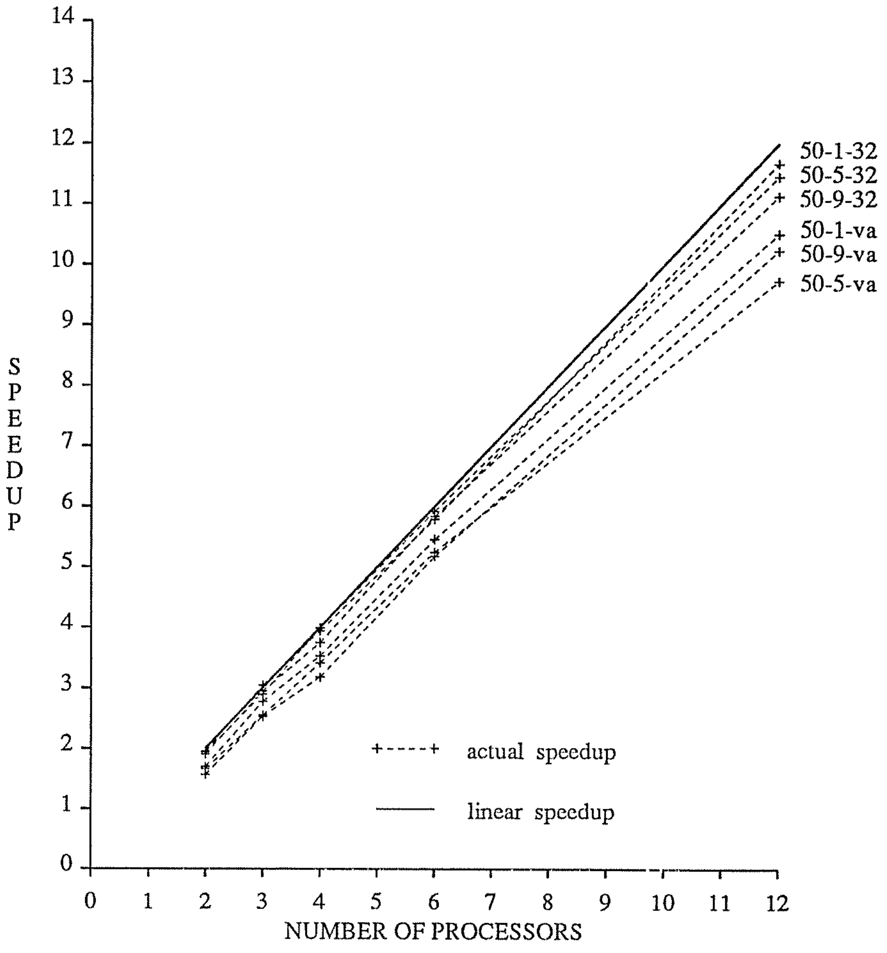

Figure 3.4 Speedup Graph for Problem Set 50

| list size = 32 / var. list size | 50-1-32 | 50-1-va | 50-5-32 | 50-5-va | 50-9-32 | 50-9-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 624 | 624 | 624 | 624 | 624 | 624 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 15918 | 15918 | 16133 | 16133 | 16324 | 16324 |
| No. pivots sequential<br>No. pivots 12 procs | 22982<br>22229 | 25418<br>23118 | 23438<br>22824 | 26035<br>23646 | 24097<br>23026 | 26501<br>24095 |
| No. cand. lists sequential<br>No. cand. lists 12 procs | 1451<br>1429 | 18938<br>11147 | 1491<br>1473 | 19914<br>11265 | 1526<br>1502 | 20832<br>11817 |
| CPU secs sequential<br>CPU secs 12 procs | 249<br>21 | 172<br>16 | 272<br>23 | 183<br>18 | 282<br>25 | 198<br>19 |
| speedup 12 procs | 11.6 | 10.5 | 11.4 | 9.7 | 11.1 | 10.2 |

Table 3.4 Multi-Period Problem Set 50

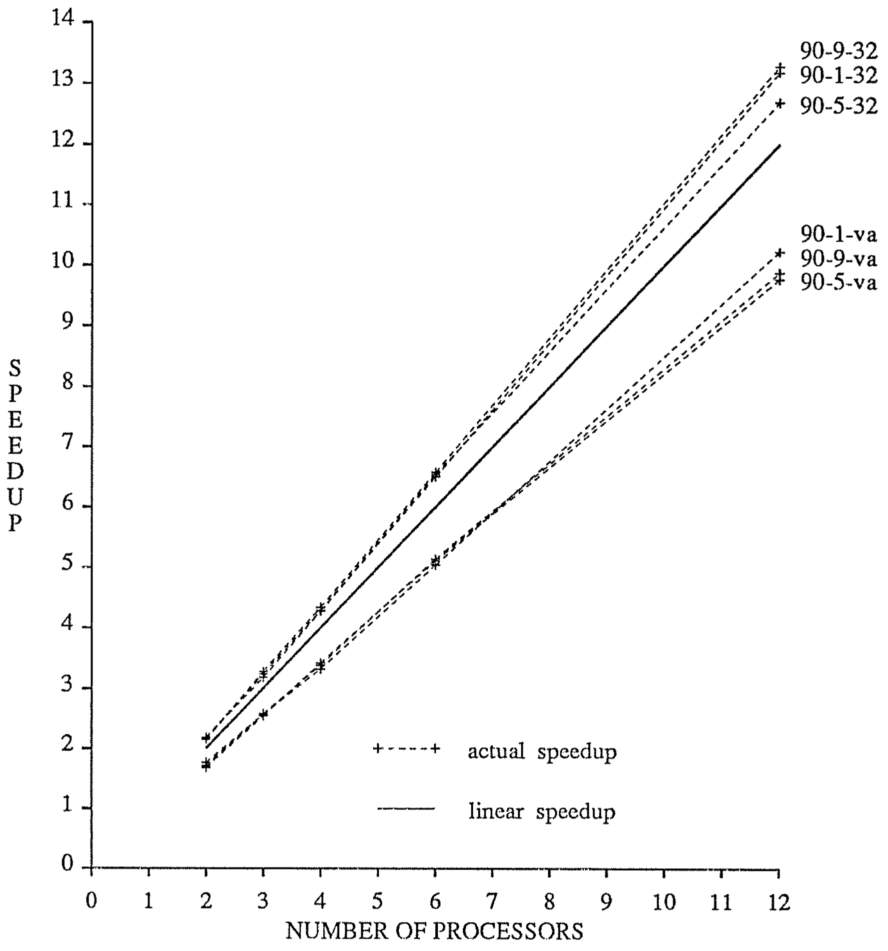

Figure 3.5 Speedup Graph for Problem Set 90

| **list size = 32 / var. list size** | **90-1-32** | **90-1-va** | **90-5-32** | **90-5-va** | **90-9-32** | **90-9-va** |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 1104 | 1104 | 1104 | 1104 | 1104 | 1104 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 16785 | 16785 | 16822 | 16822 | 16849 | 16849 |
| No. pivots sequential<br>No. pivots 12 procs | 20011<br>19165 | 21840<br>20402 | 20022<br>19146 | 22088<br>20548 | 20171<br>19148 | 22046<br>20377 |
| No. cand. lists sequential<br>No. cand. lists 12 procs | 1264<br>1225 | 21841<br>13574 | 1264<br>1228 | 22089<br>13383 | 1274<br>1230 | 22047<br>12939 |
| CPU secs sequential<br>CPU secs 12 procs | 210<br>15 | 115<br>11 | 208<br>16 | 116<br>11 | 210<br>15 | 117<br>11 |
| speedup 12 procs | 13.1 | 10.2 | 12.6 | 9.7 | 13.2 | 9.8 |

Table 3.5 Multi-Period Problem Set 90

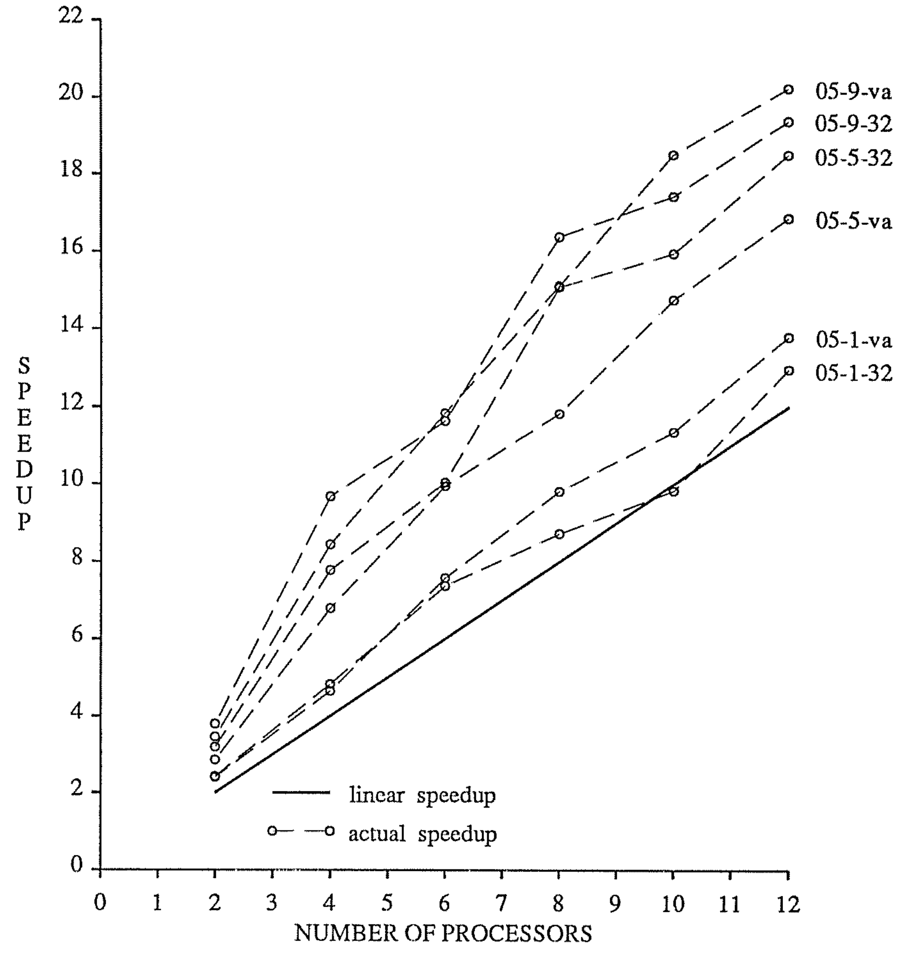

Figure 3.6 Speedup Graph for Problem Set 05

| List Size 32 / Var List Size | 05-1-32 | 05-1-va | 05-5-32 | 05-5-va | 05-9-32 | 05-9-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 84 | 84 | 84 | 84 | 84 | 84 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 14952 | 14952 | 15367 | 15367 | 15742 | 15742 |
| No. pivots sequential<br>No. pivots 12 procs | 32523<br>32960 | 33374<br>36583 | 38405<br>35809 | 38295<br>39127 | 43590<br>38366 | 42676<br>41861 |
| No. candidate lists sequential<br>No. candidate lists 12 procs | 2208<br>3223 | 13294<br>23785 | 2696<br>6266 | 13666<br>32131 | 3082<br>9671 | 12476<br>44433 |
| CPU secs sequential<br>CPU secs 12 procs | 1011<br>78 | 962<br>69 | 2160<br>116 | 1848<br>109 | 3078<br>158 | 2773<br>136 |
| speedup 12 procs | 12.9 | 13.8 | 18.5 | 16.8 | 19.4 | 20.2 |

Table 3.6 Multi-Period Problem Set 05 Solved By PGRNET

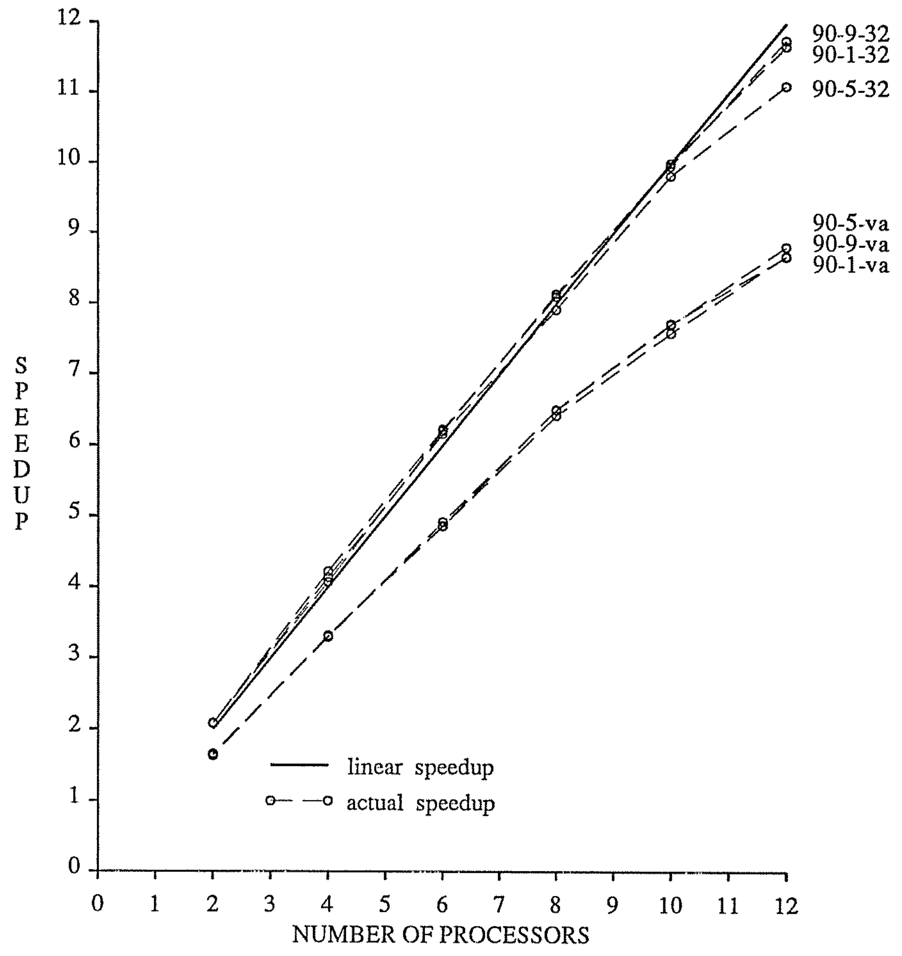

Figure 3.7 Speedup Graph for Problem Set 90

| List Size 32 / Var List Size | 90-1-32 | 90-1-va | 90-5-32 | 90-5-va | 90-9-32 | 90-9-va |
|---|---|---|---|---|---|---|
| No. qtrees at optimality | 1104 | 1104 | 1104 | 1104 | 1104 | 1104 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 16785 | 16785 | 16822 | 16822 | 16849 | 16849 |
| No. pivots sequential | 20011 | 21840 | 20022 | 22088 | 20171 | 22046 |
| No. pivots 12 procs | 19630 | 21392 | 19922 | 21467 | 19614 | 21266 |
| No. candidate lists sequential | 1264 | 21841 | 1264 | 22089 | 1274 | 22047 |
| No. candidate lists 12 procs | 1253 | 21430 | 1266 | 21467 | 1248 | 21308 |
| CPU secs sequential | 210 | 115 | 207 | 116 | 211 | 116 |
| CPU secs 12 procs | 18 | 13 | 18 | 13 | 18 | 13 |
| speedup 12 procs | 11.6 | 8.6 | 11.1 | 8.8 | 11.7 | 8.8 |

Table 3.7 Multi-Period Problem 90 Solved By PGRNET

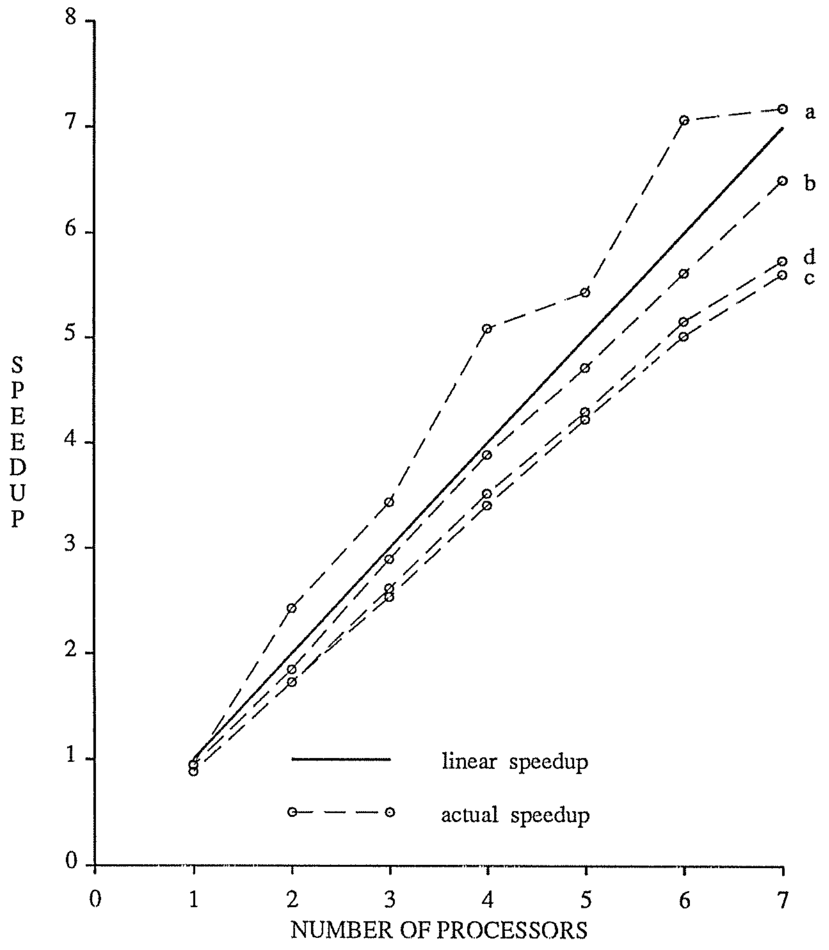

Figure 3.8 Speedup Graph for Single Period Problems

| **List Size = 32** | **a** | **b** | **c** | **d** |
|---|---|---|---|---|
| No. qtrees at optimality | 85 | 176 | 620 | 1097 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 14841 | 14932 | 15376 | 15853 |
| No. pivots sequential<br>No. pivots 7 procs | 44595<br>39190 | 34953<br>32603 | 22605<br>22676 | 19125<br>19049 |
| No. candidate lists sequential<br>No. candidate lists 7 procs | 2836<br>8047 | 2188<br>3110 | 1413<br>1481 | 1197<br>1212 |
| CPU secs sequential<br>CPU secs 7 procs | 2292<br>319 | 800<br>123 | 226<br>40 | 172<br>30 |
| speedup 7 procs | 7.1 | 6.5 | 5.6 | 5.7 |

Table 3.8 Results for Single Period Problems of Comparable Size

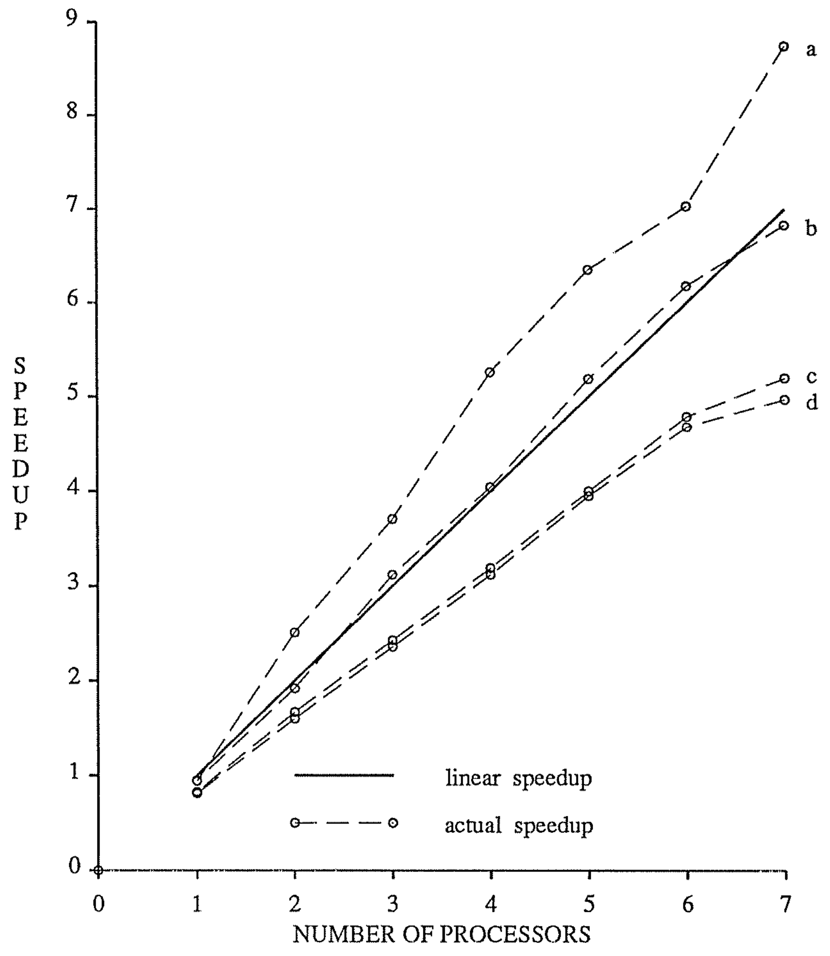

Figure 3.9 Speedup Graph for Single Period Problems

| Variable List Size | a | b | c | d |
|---|---|---|---|---|
| No. qtrees at optimality | 85 | 176 | 620 | 1097 |
| No. Nodes | 6000 | 6000 | 6000 | 6000 |
| No. Arcs | 14841 | 14932 | 15376 | 15853 |
| No. pivots sequential<br>No. pivots 7 procs | 44703<br>42552 | 36907<br>36642 | 24528<br>24674 | 20436<br>20304 |
| No. candidate lists sequential<br>No. candidate lists 7 procs | 15029<br>33738 | 17688<br>26462 | 24528<br>24874 | 20436<br>20344 |
| CPU secs sequential<br>CPU secs 7 procs | 2271<br>259 | 759<br>111 | 159<br>30 | 107<br>21 |
| speedup 7 procs | 8.7 | 6.8 | 5.2 | 4.9 |

Table 3.9 Results for Single Period Problems of Comparable Size

| **Problem Name** | **05-1** | **05-5** | **05-9** | **10-1** | **10-5** | **10-9** |
|---|---|---|---|---|---|---|
| CPU time with list size 32 | 1009 | 2159 | 3072 | 779 | 1368 | 1929 |
| CPU time with var. list size | 964 | 1848 | 2776 | 702 | 1440 | 1899 |
| Percent decrease in CPU time | 4% | 14% | 9% | 10% | ---- | 1% |

Table 3.10 Group 3, Improvement of Sequential CPU Time, subgroups 05 and 10

| **Problem Name** | **30-1** | **30-5** | **30-9** |
|---|---|---|---|
| CPU time with list size 32 | 303 | 356 | 373 |
| CPU time with var. list size | 232 | 270 | 340 |
| Percent decrease in CPU time | 23% | 24% | 8% |

Table 3.11 Group 3, Improvement of Sequential CPU Time, subgroup 30

| **Problem Name** | **50-1** | **50-5** | **50-9** | **90-1** | **90-5** | **90-9** |
|---|---|---|---|---|---|---|
| CPU time with list size 32 | 249 | 272 | 282 | 210 | 208 | 210 |
| CPU time with var. list size | 172 | 183 | 198 | 115 | 116 | 117 |
| Percent decrease in CPU time | 30% | 32% | 29% | 45% | 44% | 44% |

Table 3.12 Group 3, Improvement of Sequential CPU Time, subgroups 50 and 90

## 4. Future Directions

We have developed two algorithms for the solution of generalized network flow problems. The first, PGRNET, yields superlinear speedup results for a range of generalized network flow problems. The second yields superlinear speedup for a range of multi-period problems. Both algorithms, however, behave poorly when grain size gets quite large. In other words, both algorithms yield a poor speedup when the number of quasi-trees in the optimal basis is small.

We hope to widen the range of problems for which PGRNET performs well by developing new algorithms for large-grain problems. One strategy could involve a shared candidate list. Rather than having each processor make its own candidate list, all processors might cooperate in making a shared list, and they would take turns executing pivots. This strategy has the advantage that a sequence of pivots can be chosen which is highly optimal in the sense that all processors would contribute toward the selection of the pivot arc, rather than just one processor. No quasi-tree locking would be needed in this situation, because only one pivot would be executed at a time. The method of executing pivots one at a time would have the disadvantage that this work would not be done in parallel, even if the basis is somewhat disconnected. Another strategy that we are currently investigating involves the use of warm starts, i.e., the use of optimal basis information from a related problem to initalize the allocation of arcs to processors.

In the short term, we plan to make some simple modifications to these programs to improve their performance. We will distribute the task of creating the initial starting basis. Since this is a simple operation involving only the initialization of some arrays, this can be easily distributed between processors. We would also like to reduce the idle time associated with synchronization in STAGE 1 of MPGRNET by having the first processor to arrive there interrupt the others. Thus, no processor will sit idle during this coordination stage.

Finally, we would like to investigate the application of these strategies to pure network problems. These may be thought of as generalized networks in which the flow multiplier is -1 for all arcs. Similar tree locking or parallel pricing mechanisms may be exploited, although the locking in this case will be performed on cycles rather than quasi-trees. Preliminary computational results in this area have been promising.

## Acknowledgements

We would like to express our gratitude to the staff of the ACRF of the Argonne National Laboratory for their assistance in using their Sequent Balance 21000 for the computational results in section 3.

## References

Adolphson, D. [1982]:
"Design of primal simplex generalized network codes using a preorder thread index", Working Paper, School of Management, Brigham Young University, Provo.

Barr, R., Glover, F. and Klingman, D. [1979]:
"Enhancements of spanning tree labeling procedures for network optimization," *INFOR* 17, 16-34.

Chang, M., Engquist, M., Finkel, R. and Meyer, R. [1987]:
"A parallel algorithm for generalized networks," Technical Report # 642, Department of Computer Sciences, The University of Wisconsin-Madison (to appear in Parallel Optimization on Novel Computer Architectures, R.R. Meyer and S. Zenios, eds., Baltzer Scientific Publishing Company, 1988).

Chang, M. and Engquist, M. [1986]:
"On the number of quasi-trees in an optimal generalized network basis", *COAL Newsletter* 14, 5-9.

Engquist, M. and Chang, M. [1985]:
"New labeling procedures for the basis graph in generalized networks", *Operations Research Letters* Vol. 4, No. 4, 151-155.

Fong, L. and Srinivasan, V. [1976]:
"Multiperiod capacity expansion and shipment planning with linear costs", *Naval Research Logistics Quarterly* 23, 37-52.

Glover, F., Klingman, D. and Stutz, J. [1973]:
"Extension of the augmented predecessor index method to generalized network problems", *Transportation Science* 7, 377-384.

Glover, F., Karney, D., Klingman, D. and Napier, A. [1974]:
"A computation study on start procedures, basis change criteria, and solution algorithms for transportation problems", *Management Science* 20, 793-813.

Glover, F., Hultz, J., Klingman, D. and Stutz, J. [1984]:
"Generalized networks: a fundamental planning tool", *Management Science* 24, 1209-1220.

Hausman, W. and Gilmour, P. [1967]:
"A multi-period truck delivery problem", *Transportation Research* 14, 619-623.

Jensen, P. and Barnes J. W. [1980]:
*Network Flow Programming*, John Wiley and Sons, New York.

Kennington, J. and Helgason, R. [1980]:
*Algorithms for Network Flow Programming*, John Wiley and Sons, New York.

Klingman, D. and Mote, J. [1982]:
"A multi-period production, distribution and inventory planning model", *Advances in Management Sciences* 1, 56-76.

Mulvey J. [1978]:
"Pivoting strategies for primal-simplex network codes", *Journal of the Association for Computing Machinery* 25, 266-270.

Murtagh, B. and Saunders, M. [1978]:
"Large-scale linearly constrained optimization", *Mathematical Programming* 14, 41-72.